SECURELY COMPUTING PIECEWISE CONSTANT CODES

new arithmetic encodings for boolean functions, and applications to committed two-party computation

Benjamin E. Diamond*
Coinbase
benediamond@gmail.com

Abstract

That a maliciously secure two-party protocol is *commitment-consistent* entails that its participants, upon each execution, may supply public commitments to *each other's* inputs, which naturally bind the parties throughout the protocol. In this paper, we undertake a formal investigation of commitment-consistency, and introduce new definitions and constructions. We describe the first protocol which is *homomorphically* commitment-consistent, in that it interoperates with any homomorphic scheme (which need only be binding, with message space a prime-order group). Our protocol introduces several new techniques. We consider those boolean functions evaluated by *polynomially* many linear tests, generalizing a notion isolated by Ishai and Kushilevitz (FOCS '00); by doing so, we introduce new efficient arithmetic encodings for many boolean functions. We also describe a new protocol for secure *iterated* multiplication of secret-shared values, building upon the work of Lindell, Nof and Ranellucci (CCS '18). We describe an efficient implementation of our protocol, as well as applications in blockchains and verifiable credentials.

1 Introduction

In the classical cryptographic notion known as *commit-and-prove*, a prover simultaneously commits to a witness and proves that this witness certifies the membership of some public statement (in a given **NP** language). This notion stretches to the field's earliest days; it is implicit in the work of Goldreich, Micali and Wigderson, and is used extensively in Canetti, Lindell, Ostrovsky and Sahai [CLOS02, § 6], for example. Special instances of this notion—targeting homomorphic commitment schemes—moreover appear in many among today's most widely used zero-knowledge proof protocols, including *one-out-of-many proofs* [GK15] and *Bulletproofs* [BBB+18].

The analogous notion in secure computation has received comparatively less attention. In this latter setting, two (say) parties first obtain or generate commitments to their inputs, and then exchange these commitments. Finally, they run a secure computation, which moreover assures both parties that their respective inputs are precisely those contained in their jointly possessed commitments. This notion is extremely powerful. Jarecki and Shmatikov [JS07, § 1], for example, observe that a "secure *committed 2PC* protocol is a much more useful tool than a standard 2PC protocol", as 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."

We describe various concrete applications of commitment-consistency in Subsection 1.5 below; we briefly survey these now. Commitment-consistent two-party computation facilitates, for example, a much-stronger variant of *Yao's Millionaires' problem*, whereby the millionaires' balances reside *not* just in their respective heads—where they may be trivially falsified—but rather in an ongoing, live-deployed private payment protocol (such as Monero [NMt16] or Anonymous Zether [Dia21]), and in which the output of the protocol necessarily reflects these real hidden balances. It also allows users of *anonymous credentials* (see e.g. [CDL16]) to securely compute upon the values endorsed within their credentials (while assuring each other of consistency with these credentials). Informally, commitment-consistency allows parties to *force each other to use the "right" input.*

*I would like to thank Yehuda Lindell, Jason Long, Samuel Ranellucci, and Amir Yehudayoff for extremely helpful discussions.

In this work, we describe the first maliciously secure, commitment-consistent two-party protocol which works with *any* homomorphic commitment scheme of prime order. Crucially, our protocol accepts as inputs explicit, *previously generated* commitments, and assures consistency with them. (It also supports *ciphertext*-consistent computation.) Our protocol assumes only the existence of prime-order homomorphic encryption, and can be instantiated under the DDH assumption alone. We also introduce new definitional contributions for commitment-consistent secure computation. Indeed, following the spirit of [CLOS02, Fig. 4], we define the *commitment-consistent computation* functionality as that which accepts the parties' inputs during its commitment phase, and computes upon them during its evaluation phase (see Definition 2.8 below).

Our core technical contribution is two-fold. On the one hand, we develop a family of novel boolean-to-arithmetic encodings; on the other, we describe a concretely efficient protocol for two-party, *commitment-consistent* secure iterated modular multiplication. These contributions directly complement each other; indeed, our encoding technique yields arithmetic circuits (of a particular form) upon which our multiplication protocol operates especially efficiently. We briefly summarize each of these contributions.

Targeting the application of arithmetic techniques to boolean function evaluation, we introduce a new boolean-to-arithmetic encoding paradigm based on *exact hyperplane covers*. For many functions, the circuits we obtain by this means improve upon the complexity attained by existing known boolean-to-arithmetic encoding techniques; we mention in particular the classical encoding technique (see e.g. Cramer, Damgård and Nielsen [CDN15, Ex. 3.1]) and the more advanced construction of Ishai and Kushilevitz [IK00].[1] We emphasize that this advance yields gains *regardless* of how the resulting arithmetic circuit is evaluated. Indeed, though we indeed ultimately propose a particular mechanism, other mechanisms could just as well be used; we mention for example the standard *BGW-type* family of approaches, described e.g. in Cramer, Damgård and Nielsen [CDN15, S 3.3] and in Asharov and Lindell [AL17], as well as *SPDZ-type* techniques [DKL$^+$13] (these approaches would sacrifice commitment-consistency, however). For example, our hyperplane-based encoding for the *bitwise equality function*—see Example 3.29—yields a *constant-depth, linearly-sized* arithmetic encoding of that function; the classical technique [CDN15, Ex. 3.1] yields a logarithmic-depth circuit in the best case, while [IK00] yields a constant-depth, but quadratically-sized, arithmetic circuit.[2]

Informally, the key insight of our approach is that we exploit *size of the field* in a novel way. Both [CDN15, Ex. 3.1] and [IK00] operate essentially independently of the field's characteristic q; our approach requires that q grow exponentially in n. On the one hand, this mild restriction entails that the parties' inputs be relatively "small" (or that the field be large); on the other, it yields smaller and more efficient circuits. To illustrate this, we continue with the example of the equality function of Example 3.29. For this function, the input-size restriction entails that the parties' bitstrings be of length less than $\log_3 2 \cdot (\log_2 q + 1)$; for standard fields, for which $\lceil \log_2 q \rceil = 256$ say, this entails that the parties' *individual* bitstrings each be of length at most $\frac{n}{2} \leq 162$ bits (actually, an improvement to $\frac{n}{2} \leq \lceil \log_2 q \rceil = 256$ bits is possible, but we omit this, for expository purposes). In exchange for this length restriction, one obtains a constant-depth, linearly-sized encoding. No arithmetic encoding of this function matching this efficiency has been previously reported.

Indeed, our encoding for the function f_n of Example 3.29 is quite unusual. Among other things, it demonstrates that, by evaluating *just one* $\mathbb{F}_q$-linear function over n input bits, one can determine whether the input string's even-indexed and odd-indexed $\frac{n}{2}$-bit substrings are equal. Roughly, on input $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$, our encoding proceeds by interpreting the $\frac{n}{2}$ *differences* $(x_{2i} - x_{2i+1})_{i=0}^{n/2-1}$—each taking values in $\{-1, 0, 1\}$—as the *trits* of an integer encoded in length-$\frac{n}{2}$ *balanced ternary*; it then computes a natural powers-of-3 combination of these trits. Crucially, if q is sufficiently large—specifically, if $q > \frac{3^{n/2}}{2}$—then this combination is guaranteed *not* to overflow or underflow modulo q, and so equals 0 precisely when the two bitstrings are equal, as desired. Finally, we must further multiply the resulting value by a random $\mathbb{F}_q$-element, in order to preserve privacy; indeed, after doing so, we obtain a scalar which is *either* 0 or random—depending on whether $f_n(\mathbf{x})$ is 1 or 0—and so reveals *only* $f_n(\mathbf{x})$ (and nothing further about $\mathbf{x}$). The resulting circuit thus consists *only* of linear operations, followed by a *single* field multiplication.

[1] A *arithmetic encoding* over $\mathbb{F}_q$ of a boolean function $f_n : \{0,1\}^n \to \{0,1\}$ is an arithmetic circuit C_n over $\mathbb{F}_q$, for which, for each boolean input $\mathbf{x} \in \{0,1\}^n$, $C_n(\mathbf{x}) = f_n(\mathbf{x})$. We actually target a certain relaxation whereby C_n may have *random* inputs, and, for each $\mathbf{x} \in \{0,1\}^n$, the output *distribution* $C_n(\mathbf{x})$ depends only on $f(\mathbf{x})$. We refer to [IK00, § 2.1] for further discussion.

[2] Indeed, any boolean circuit evaluating $f_n : (x_0, \ldots, x_{n-1}) \mapsto \bigwedge_{i=0}^{n/2-1} (x_{2i} \oplus \overline{x_{2i+1}})$ must have depth at least $\lceil \log n \rceil$ (this elementary fact follows from e.g. Vollmer [Vol99, Ex. 3.1]), and the transformation [CDN15, Ex. 3.1] is depth-preserving. The transformation of [IK00] induces a constant-depth circuit whose size grows quadratically in f_n's *nondeterministic mod-q branching program complexity*; this latter complexity measure grows as $\Omega(n)$ on the bitwise equality function f_n.

We further note the following geometric interpretation of our above construction. Indeed, it consists of no more and no less than the construction of a *hyperplane* $H \subset \mathbb{F}_q^n$—specifically, that defined by $\sum_{i=0}^{n/2-1} 3^i \cdot (x_{2i} - x_{2i+1}) \equiv 0 \pmod{q}$—whose *intersection* with the discrete unit cube $\{0,1\}^n \subset \mathbb{F}_q^n$ consists precisely of those bitstrings $\mathbf{x} \in \{0,1\}^n$ for which $f_n(\mathbf{x}) = 1$. In fact, this *generic condition* has been identified before; it appears precisely as [IK00, Def. 5.4 1.], for example. Our first observation, then, is that the class of boolean functions which "test a linear condition" over $\mathbb{F}_q$ is more interesting than previously known (at least if q is assumed large); indeed, it contains useful boolean functions.

More generally, however, we introduce the consideration of boolean functions evaluated by *a polynomial number* of linear tests (as opposed to just one); this directly generalizes Ishai and Kushilevitz [IK00, Def. 5.4 1.]. Indeed, our encoding technique, geometrically speaking, consists of *exactly covering* the "on-set" or "off-set" $S \subset \{0,1\}^n$ of some particular boolean function (that is, the set of input bitstrings upon which it evaluates to *true* or *false*, respectively) with a polynomial number of affine hyperplanes over a large prime field $\mathbb{F}_q$. The encodings we obtain in this way often improve the complexity of previously known encodings. Indeed, as a further example, we represent the *unsigned integer comparison* function f_n on two $\frac{n}{2}$-bit unsigned integers as a collection of $\frac{n}{2}$ $\mathbb{F}_q$-affine-linear tests (see Example 3.30). Our representation's correctness requires that $q \geq \frac{3}{2} \cdot 2^{n/2}$. Our representation in turn yields an arithmetic encoding of f_n of $O(\log n)$ depth, which uses only $\frac{n}{2}$ field multiplications. This significantly surpasses—albeit not asymptotically in this case—the efficiency of [CDN15, Ex. 3.1] (even assuming that this latter method is applied to a log-depth boolean comparator circuit). It also asymptotically improves upon [IK00] in size, though not in depth.

Slightly more formally, we isolate two key new complexity classes—which we call **H** and **co-H**—consisting of those boolean functions whose *on-sets* and *off-sets* (respectively) admit exact coverings by polynomial-cardinality collections of affine hyperplanes (over an appropriately sized prime field). These definitions represent the first use within cryptography of *exact hyperplane covers*, which, in fact, are classically studied within algebraic combinatorics (see e.g. [AF93] and [AGG+21]; we refer to 1.1 for further discussion). We prove that these classes are large and expressive, by relating them to a generic family of cube subsets known as *piecewise constant codes* (see Subsections 1.2 and 3.1). Using this result, we devise many examples and constructions; the further concrete examples we treat include those functions decided by depth-two circuits (Theorem 3.14), symmetric functions (Theorem 3.24), assessors of the disjointness of sets (Example 3.15), the indicator functions of piecewise constant codes (Example 3.31), and more. Our constructions, in general, are new and striking. We finally undertake a thorough complexity-theoretic investigation of **H** and **co-H**, and relate them to known classes (we survey and present these results in Subsections 1.1 and 3.2, respectively).

Our second major contribution consists of a *cryptographic* protocol for the secure evaluation of functions in **H** or **co-H**. Our key subprotocol is a *commitment-consistent, iterated* modular multiplication technique for secret-shared values. This protocol is interesting in its own right; it consists of a subtle, recursive extension of the *two-argument* multiplication protocol given in Lindell, Nof and Ranellucci [LNR18, § 6.1]. Indeed, our adaptation—compared to the naïve approach, in which that protocol is carried out repeatedly in a tree-like manner—cuts roughly in half the number of times which certain among its key subfunctionalities must be invoked. We summarize this protocol in Subsection 1.3 and describe it in full in Subsection 4.2.

1.1 Hyperplane coverings in mathematics and cryptography

We now describe in slightly more detail our definition and investigation of the complexity classes **H** and **co-H**. Purely for notational convenience, we let n be even in what follows. An *intersection pattern* between a hyperplane and the discrete unit cube is a subset $C \subset \{0,1\}^n$ of the form $C = \{0,1\}^n \cap H$, where $H \subset \mathbb{F}_q^n$ is an affine hyperplane (we only consider hyperplanes over prime fields in this work). The study of intersection patterns goes back at least to Littlewood and Offord [LO43]. Indeed, the classical *Littlewood–Offord problem* asks, essentially, how *large* an intersection pattern $H \cap \{0,1\}^n$ can be, specifically under the assumption that H's coefficients are *nonzero*. (If 0 coefficients are allowed, then the large intersection $|H \cap \{0,1\}^n| = 2^{n-1}$ may be trivially attained, by the hyperplane $x_0 = 1$, for example.) Treating a finite-field analogue of the Littlewood–Offord problem (the original problem was posed over $\mathbb{C}$), Griggs [Gri93, Cor. 1], in relatively recent work, shows that any affine hyperplane $H \subset \mathbb{F}_q^n$ with nonzero coefficients necessarily satisfies $|H \cap \{0,1\}^n| \leq \binom{n}{n/2} = \Theta\left(\frac{2^n}{\sqrt{n}}\right)$ (provided that q is not too small). This tighter bound is, in fact, the best possible, attained for example by $\sum_{i=0}^{n-1} x_i = \frac{n}{2}$.

A cube subset $S \subset \{0,1\}^n$ is said to be *exactly covered* by hyperplanes $\{H_i\}_{i=0}^{m-1}$ over $\mathbb{F}_q$ if $S = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$ (the latter intersection is taken within $\mathbb{F}_q^n$). The problem of exactly covering cube subsets by means of collections of hyperplanes has appeared throughout a handful of prior works. The covering of a cube subset by a *single* hyperplane appears implicitly in [IK00, Def. 5.4 1.]. While the entire cube may be trivially covered by two hyperplanes, a seminal result of Alon and Füredi [AF93, Thm. 4] shows that *no fewer than* n hyperplanes can exactly cover the cube *minus* one element (that is, the set $S = \{0,1\}^n - \{(0,\ldots,0)\}$, say). As n hyperplanes clearly suffice, this result is sharp (indeed, one may use the hyperplanes $x_i = 1$, for $i \in \{0,\ldots,n-1\}$). Aaronson, Groenland, Grzesik, Johnston, and Kielak [AGG$^+$21] study 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. They show nonconstructively that this number is exponential in n. Diamond and Yehudayoff [DY22] describe a certain *concrete* cube subset $S \subset \{0,1\}^n$ which can be covered only using exponentially many hyperplanes. This set is precisely that corresponding to pairs of *disjoint* subsets of $\{0,\ldots,\frac{n}{2}-1\}$; interestingly, this latter set appears in many celebrated works in communication complexity (see e.g. Razborov [Raz92] and Rao and Yehudayoff [RY20]).

Our work is the first to investigate exact coverability by *polynomially*-sized collections of hyperplanes (in the dimension n); we are also the first to relate exact coverability by hyperplanes to *computation* (of any sort). Indeed, our class **H**, roughly, consists of those languages $L \subset \{0,1\}^*$ for which each intersection $L \cap \{0,1\}^n$ admits a *polynomial*-cardinality covering by hyperplanes over $\mathbb{F}_q$ (where q is some n-bit prime); we write **co-H** for **H**'s language-wise complement. (The relationship between **H** and **co-H** is analogous to that between **NP** and **co-NP**; we give further details in Section 3 below.) Beyond its direct cryptographic import, our isolation of the classes **H** and **co-H** seems to be of general theoretical significance. Indeed, the exact characterization of **H** and **co-H** is an extraordinarily difficult problem; in a series of mathematical results, we achieve a rather complete characterization of these classes, as they relate to more classical classes in circuit complexity. We show, for example, that $\mathbf{\Sigma_2} \subset \mathbf{H}$ and $\mathbf{\Pi_2} \subset \mathbf{co\text{-}H}$ (Theorem 3.14), as well as that $\mathbf{H} \not\subset \mathbf{AC}^0$ and $\mathbf{co\text{-}H} \not\subset \mathbf{AC}^0$ (Corollary 3.27). We also present various negative containment results on these classes. Exploiting the work of Diamond and Yehudayoff [DY22], we establish the *non-inclusions* $\mathbf{\Pi_2} \not\subset \mathbf{H}$ and $\mathbf{\Sigma_2} \not\subset \mathbf{co\text{-}H}$ (see Theorem 3.18 below); we derive as corollaries the further non-inclusions $\mathbf{AC}^0 \not\subset \mathbf{H}$ and $\mathbf{AC}^0 \not\subset \mathbf{co\text{-}H}$ (see Corollary 3.20). We further conclude that $\mathbf{H} \neq \mathbf{co\text{-}H}$ (see Corollary 3.21); this difficult result is interesting, in light of its rough resemblance to the conjectured relationship $\mathbf{NP} \stackrel{?}{\neq} \mathbf{co\text{-}NP}$. Finally, we prove the upper containments $\mathbf{H} \subset \mathbf{NC}^1$ and $\mathbf{co\text{-}H} \subset \mathbf{NC}^1$ (see Theorem 3.32).

1.2 Hyperplane coverings from piecewise constant codes

Our main technical tool for our study of **H** and **co-H** consists of a key sequence of mathematical lemmas (see Subsection 3.1) which relate coverability by hyperplanes to a certain coding-theoretic construction. Interestingly, our hyperplane constructions sketched above all arise as special cases of this general result.

We explain our approach as follows. *Piecewise constant codes* (see e.g. [CHLL97, § 3.3]) constitute a large and tractable class of codes, with a rich combinatorial structure. Each such code $S \subset \{0,1\}^n$ is expressed with the aid of a positive integer partition $n = n_0 + \cdots + n_{t-1}$ (in the number-theoretic sense) 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"—that is, selecting—certain cells within this array. Our core mathematical result shows that many piecewise constant codes admit polynomial-cardinality hyperplane coverings. To show this, we isolate a certain key structure within the multidimensional array associated to a piecewise constant code, which we call a "quasicube". In a central lemma (Lemma 3.9 below), we show that the cube 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 (indeed, it's enough that q have n bits). It follows immediately (see Theorem 3.12) 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 sharp, in the sense that subsets $S \subset \{0,1\}^n$ which *don't* admit compact representations of this form also lack efficient hyperplane coverings (see Example 3.16, Lemma 3.17 and Theorem 3.18).

1.3 From hyperplane coverings to committed two-party computation

We now survey in slightly further detail our cryptographic techniques. We introduce several new cryptographic protocols, which, in conjunction, serve to securely compute with commitment-consistency *any* set $S \subset \{0,1\}^n$ expressed as a union $S = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$ of intersection patterns. Our protocol requires $O(\log m)$ rounds, $O(n \cdot m)$ computation, and $O(m)$ communication; in particular, it is efficient when m grows polynomially in n. Our protocol involves the use of public-key primitives of prime order q; it can be securely instantiated under the DDH assumption alone. (Though our techniques appear to generalize readily to the multiparty setting, we refrain from treating it explicitly in this work.)

We sketch now the rough idea of our main protocol, which we specify explicitly in Protocol 4.18. Roughly speaking, the parties P_0 and P_1, given an even integer n, a function $f_n : \{0,1\}^n \to \{0,1\}$, and a hyperplane covering $f_n^{-1}(1) = \cup_{i=0}^{m-1} H_i \cap \{0,1\}^n$ (say), on inputs $\mathbf{x}_0$ and $\mathbf{x}_1$ in $\{0,1\}^{n/2}$, 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; each party simultaneously proves, using a standard protocol, that these latter bits give exactly the binary representation of an appropriate prior, committed value.

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 sequence of hyperplanes $H_0, \ldots, H_{m-1}$ covering $f_n^{-1}(1)$, both on their plaintext shares and, simultaneously, on their ciphertexts representing the other party's shares. In this way, they obtain respective additive sharings of the outputs under the hyperplanes $\{H_i\}_{i=0}^{m-1}$ of their joint input, 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 $f_n^{-1}(1) = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$, in light of the homomorphic properties of hyperplane evaluation, now implies that $f_n(\mathbf{x}) = 1$ if and only if *at least one* of the parties' now-held additive 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 simply securely multiply their m additively shared outputs. If the parties moreover securely multiply the resulting product with a further shared random scalar, the resulting product is then *either* 0 or uniformly random, depending on $f_n(\mathbf{x})$; the parties thus learn the result $f_n(\mathbf{x})$ (and nothing more). We thus turn to the problem of secure, commitment-consistent iterated modular multiplication.

Interestingly, *two-multiplicand* secure two-party multiplication has received extensive recent attention, in connection with threshold ECDSA signing; indeed, it is precisely the central task of Lindell, Nof and Ranellucci [LNR18, § 6.1]. In that paper, an "underlying" private multiplication protocol—like that of Doerner, Lee, Kondi and shelat [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.

Our protocol introduces interesting new techniques, which target the setting in which the parties must conduct $\Theta(m)$ secure multiplications (as opposed to a constant number). Indeed, we propose a tree-like, recursive variant of [LNR18, Prot. 4.7], and describe several new optimizations. In each round, the parties begin by vectorizing a number of underlying multiplications, handling a single layer of the tree. Moreover, 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.

1.4 Comparison to prior work in commitment-consistency

The *explicit* treatment of commitment-consistent secure computation is the literature is rather sparse. Lindell and Rabin [LR17], for example, 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." Our goal in this paper is to study *explicit* input-consistency with commitments; specifically, it's crucial that the commitments be generated externally, and supplied as inputs to the protocol. In particular, it does not suffice if the inputs are "committed" only implicitly, at some point *during* the course of the execution of the protocol.

Our protocol is decidedly *less* concretely efficient than, say, garbling-based approaches (we note, for example, that of Katz, Ranellucci, Rosulek, and Wang [KRRW18]). Those approaches, however, do *not* yield commitment-consistency, and so would represent an "unfair" comparison. Indeed, we do not claim to best the concrete efficiency of two-party malicious protocols which *don't* guarantee commitment-consistency.

One may, in theory, obtain commitment-consistency generically, by encoding some *particular* scheme's commitment function within an explicit boolean circuit, and applying a generic technique for two-party boolean circuit evaluation. While this approach is presumably suitable for commitment schemes based on symmetric constructions, it's unfeasible for typical homomorphic commitment schemes (like the Pedersen scheme), whose commitment functions invoke public-key operations. Indeed, elliptic curve scalar multiplication is far too complex to encode in a "flattened" boolean circuit (i.e., one represented as a flat list of gates, with no reusable submodules or sequential logic)[3].

In certain garbling schemes—specifically, in ones in which the evaluator conveys its inputs using an elliptic-curve-based instantiation of oblivious transfer—it may be possible to bind the *evaluator's* inputs to public commitments (at least under certain *particular* schemes, like Pedersen's). It is not obvious how the *garbler's* inputs could be bound to commitments. We emphasize that our protocol is "bilaterally" commitment-consistent, in that both parties' inputs are symmetrically bound to commitments.

Interestingly, a work of Jarecki and Shmatikov [JS07] explicitly targets commitment-consistent, maliciously secure two-party computation. That protocol, however, works only with a particular, Camenisch–Shoup-style commitment scheme, itself based on Paillier-like groups of unknown order. Our protocol works over *prime-order* groups, and allows arbitrary such commitment schemes (our protocol treats its commitment scheme in a black-box manner, and requires only that its commitment function be an $\mathbb{F}_q$-homomorphism).

Frederiksen, Pinkas and Yanai [FPY18] present a "compiler" which bootstraps any commitment scheme of prime order into a secure multiparty protocol for arithmetic circuits. Their protocol does not obviously interoperate with *externally* supplied commitments; indeed, it begins with commitments to *random* values, which are unrelated to prior reference commitments. It may be possible that their work could be adapted to this purpose, but this capability is not made explicit. Their protocol targets only *arithmetic* circuits, and does not support the evaluation of boolean functions, though it perhaps could be made to do so (possibly with the aid of the "bit proofs" of Groth and Kohlweiss [GK15, Fig. 1]). If it were to be used in this way, then that protocol could in fact be used directly on our new boolean-to-arithmetic encodings, instead of on classical encodings, and in doing so accrue corresponding efficiency gains (though without commitment-consistency). Finally, their protocol is not implemented, and its concrete efficiency seems uncertain.

We mention finally the "SPDZ-type" family of protocols for secure arithmetic circuit evaluation (see for example Damgård, Keller, Larraia, Pastro, Scholl, and Smart [DKL$^+$13]). While these protocols, again, likely beat ours in concrete efficiency, they do *not* offer commitment-consistency, and so would make for an unfair comparison. Finally (like [FPY18]), the SPDZ protocols target arithmetic circuits. Though they could *possibly* be adapted to the boolean and commitment-consistent setting, this adaptation would necessarily mandate the use of new zero-knowledge proofs (for commitment-consistency at each input wire, as well as to prove that the inputs are indeed bits [GK15, Fig. 1]). This transformation would thus introduce public-key operations at each input wire, and thereby void the efficiency of these protocols (which itself stems from from their use of information-theoretic MACs). It is possible that the resulting protocol would be comparable in efficiency to ours, but it is not clear; it would moreover require preprocessing, which our protocol does not. In any case, that protocol could also be used on our new boolean-to-arithmetic encodings to great effect.

[3] We conducted an interesting experiment in this direction. We were able to fully specify the Pedersen scheme over the binary Koblitz curve *K-233* using purely combinational logic in Verilog. Our attempt to *flatten* this circuit using the open-source synthesis tool *Yosys* was prohibitively memory-intensive, and exhausted the memory of a cloud machine with 384 GB of RAM.

In general, our task is to demonstrate the theoretical and practical feasibility of explicit commitment-consistency; we intend for our work to be only the first in this important branch of research.

1.5 Applications of commitment-consistency

We now survey possible applications of commitment-consistency. We argue that commitment-consistency is a crucial ingredient in *meaningful* real-life secure computation. Indeed, in most practical applications of secure computation, it's not enough that the protocol to be "secure"; in addition, the parties must use the "right" input. We describe real-life examples of this phenomenon below.

1.5.1 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 dispersed across UTXOs). As an important special case, two parties could, for example, *compare* their live balances within an ongoing private payment protocol. This capability yields a much-stronger variant of *Yao's Millionaires' problem*, in which the two millionaires' balances are *real* (and cannot be falsified). Indeed, the boolean integer comparison function has an efficient hyperplane covering (see Example 3.30 below).

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

1.5.2 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.

1.6 Concrete efficiency

Our protocol is concretely implemented, and is practical. In Subsection 4.4, we describe a full implementation of our protocol. For the sake of our benchmarks, we specialize f_n to the integer comparator function (see Example 3.30). Our protocol is practical, and runs over a WAN in about as much time as a private cryptocurrency transaction takes to generate (see e.g. [Dia21] for an overview). Specifically, on the function $f_{64} : \{0,1\}^{64} \to \{0,1\}$ 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 overhead is inherited from the multiplication subprotocol [DKLs18, Prot. 8]. We give further details in Subsection 4.4.

1.7 A further prior work

We mention a further important progenitor of our work, 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 characterize their protocol in these terms, their method, in fact, 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 directly and reports whether a zero is present. Their protocol lacks malicious security, and requires a third party; moreover, it treats only one function.

In any case, their approach is interesting, and anticipates a handful of fundamental features of our work. For example, they notice that their *particular* hyperplane covering requires only the mild inequality $q > n$; indeed, they set $n = 64$, and use the unusually small prime $q = 67$. This works because their particular on-set is actually a *subcube* (see Subsection 2.3 for definitions); for this simple class of cube subsets, hyperplane coverings exist even for small q (we discuss this in Subsection 3.2; see e.g. Theorem 3.14). In general, the functions we treat require much larger q, say in the range $\{2^{n-1}, \ldots, 2^n - 1\}$ (see Lemma 3.9).

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 a non-degenerate affine-linear functional $H : \mathbb{F}_q^n \to \mathbb{F}_q$ (see Cohn [Coh82, § 4.8]). Each hyperplane admits an expression $H : (x_0, \ldots, x_{n-1}) \mapsto a_0 \cdot x_0 + \cdots + a_{n-1} \cdot x_{n-1} - v$, for appropriate field elements $a_0, \ldots, a_{n-1}, v$ 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 $\mathbb{F}_q$-affine-linear map $o_{\mathbf{x}} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ which maps $\{0,1\}^n$ to itself and 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:

$$o_{\mathbf{x}}(\mathbf{y}) : (y_0, \ldots, y_{n-1}) \mapsto \left(\begin{cases} 1 - y_i & \text{if } x_i = 1 \\ y_i & \text{if } x_i = 0 \end{cases} \right)_{i=0}^{n-1}$$

clearly satisfies the desired properties. □

We note that, on the unit cube itself, $o_{\mathbf{x}}$ restricts to the XOR-by-$\mathbf{x}$ map.

2.2 Boolean function complexity

We refer to Wegener [Weg87] and Vollmer [Vol99] for facts about the complexity of boolean functions. A *family of sets* takes the form $\{S_n \subset \{0,1\}^n\}_{n\in\mathbb{N}}$. There is an obvious natural correspondence between families of sets and families of boolean functions $\{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ (effected by associating each boolean function to its on-set and each on-set to its indicator function). We occasionally speak of these two objects interchangeably. The more classical notion of a *language* likewise arises equivalently; to each $L \subset \{0,1\}^*$, we associate the family of sets $\{S_n := L \cap \{0,1\}^n\}_{n\in\mathbb{N}}$. In our below treatment, we refer only to families of sets and functions, and not to languages; the notions are nonetheless equivalent.

For each natural number k, $\mathbf{\Sigma_k}$ and $\mathbf{\Pi_k}$ denote the set families decided by polynomially-sized, unbounded fan-in, layered circuits with an OR or an AND gate at the output (respectively) and k alternating layers of gates subsequently, and with negations only applied to the inputs (see [Weg87, § 11 Def. 1.1]). By De Morgan's laws, the classes $\mathbf{\Sigma_k}$ and $\mathbf{\Pi_k}$ are element-wise complements, for each k. 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 set 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'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 sorted 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 $\mathbf{x}_0 \,\|\, \cdots \,\|\, \mathbf{x}_{t-1}$ 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 $\mathbf{x}_0 \,\|\, \cdots \,\|\, \mathbf{x}_{t-1}$ 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 $\mathbf{x}_0 \,\|\, \cdots \,\|\, \mathbf{x}_{t-1} \in \{0,1\}^n$ 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 $\widehat{S} \subset \prod_{i=0}^{t-1}\{0,\ldots,n_i\}$ consisting of those multi-indices $(w_0,\ldots,w_{t-1})$ for which S contains any, and hence every, word $\mathbf{x}_0 \,\|\, \cdots \,\|\, \mathbf{x}_{t-1} \in \{0,1\}^n$ 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 $\sum_{(w_i)_{i=0}^{t-1}\in\widehat{S}} \prod_{i=0}^{t-1}\binom{n_i}{w_i}$. The covering radius of a piecewise constant code $S \subset \{0,1\}^n$ is exactly the "covering radius" of $\widehat{S} \subset \prod_{i=0}^{t-1}\{0,\ldots,n_i\}$, 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 $\widehat{C} \subset \prod_{i=0}^{t-1}\{0,\ldots,n_i\}$. The *pullback* $C \subset \{0,1\}^n$ of $\widehat{C}$ is defined by $C := \left\{ \mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1} \in \{0,1\} \;\middle|\; (w(\mathbf{x}_0),\ldots,w(\mathbf{x}_{t-1})) \in \widehat{C} \right\}$.

Informally, the pullback $C \subset \{0,1\}^n$ is defined to be the union, over those cells $(w_0,\ldots,w_{t-1}) \in \widehat{C}$, 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 $\lambda \in \mathbb{N}$,

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

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 $\lambda \in \mathbb{N}$,

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

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* $\mathsf{PubK}^{\mathsf{LR-cpa}}_{\Pi,\mathcal{A}}(\lambda)$ 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 $\Pr[\mathsf{PubK}^{\mathsf{LR-cpa}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + \mu(\lambda)$.

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 $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$, 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 subfunctionality 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

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

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 $f_n : \{0,1\}^n \to \{0,1\}$. 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 $f_n : \{0,1\}^n \to \{0,1\}$, 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 $v := f_n(\mathbf{x})$, 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. 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. □

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 $\phi : x \mapsto x \cdot g$. We record further applications of Theorem 2.14 here; we will use these below.

Example 2.15. 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 2.16. 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 2.17. 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 2.18. Protocol 2.13 can be used to prove that two ciphertexts encrypt the same message. We have the relation:

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

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 2.19. 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 2.20. 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 2.21. 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 2.22 (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 NIZK" ideal functionality $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ (see e.g. [LNR18, Func. 3.4]):

FUNCTIONALITY 2.23 ($\mathcal{F}^R_{\mathsf{com\text{-}zk}}$—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, $\mathcal{F}^R_{\mathsf{zk}}$ 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, $\mathcal{F}^R_{\mathsf{com\text{-}zk}}$ checks whether $(\texttt{commit-prove}, x, R(x, w))$ is in memory. If it is, $\mathcal{F}^R_{\mathsf{com\text{-}zk}}$ sends $(\texttt{decommit-prove}, x, R(x, w))$ to $P_{1-\nu}$; otherwise, $\mathcal{F}^R_{\mathsf{com\text{-}zk}}$ sends $(\texttt{decommit-prove}, x, 0)$ to $P_{1-\nu}$.

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

3 Piecewise Constant Codes and Hyperplane Coverings

In this section, we study which boolean functions $f_n : \{0,1\}^n \to \{0,1\}$—or more precisely, which sets $S_n \subset \{0,1\}^n$—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 $S_n \subset \{0,1\}^n$ if $S_n = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$.

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

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

That is, $\{S_n\}_{n\in\mathbb{N}}$ is in **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 $S_n = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$, where each $H_i \subset \mathbb{F}_q^n$ is an affine hyperplane. In this case, we also say that the family $\{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ defined by $f_n(\mathbf{x}) := \mathbf{x} \in S_n$ is in **H**.

Definition 3.3. The class **co-H** consists of those families $\{S_n \subset \{0,1\}^n\}_{n\in\mathbb{N}}$ for which, for each $n \in \mathbb{N}$, the complement $\overline{S_n} \subset \{0,1\}^n$ can be covered by polynomially many hyperplanes over some fixed n-bit prime q.

The family $\{S_n\}_{n\in\mathbb{N}}$ is in **co-H** if and only if $\left\{\overline{S_n}\right\}_{n\in\mathbb{N}}$ is in **H**. In this case, we also say that the family $\{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ defined by $f_n(\mathbf{x}) := \mathbf{x} \in S_n$ is in **co-H**.

3.1 Main theorem on piecewise constant codes and intersection patterns

Our main mathematical result shows that those sets $S_n \subset \{0,1\}^n$ expressible as "compact" piecewise constant codes are also coverable by polynomial-cardinality collections of hyperplanes.

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

Lemma 3.4. *We 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 $n_i + 1$. □

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

Lemma 3.5. *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 3.4 and 3.5 evoke various classical problems (see e.g. Došlić [Došl05]).

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

Definition 3.6. Fix $n \in \mathbb{N}$ and a partition $n = n_0 + \cdots + n_{t-1}$. We call a subset $\widehat{C} \subset \prod_{i=0}^{t-1}\{0,\ldots,n_i\}$ a *quasicube* if $\widehat{C}$ takes the form

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

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 3.7. 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 3.8. 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 $\widehat{S} = S$ 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 3.9. *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 $\widehat{C} \subset \prod_{i=0}^{t-1}\{0,\ldots,n_i\}$, the pullback $C \subset \{0,1\}^n$ of $\widehat{C}$ is an $\mathbb{F}_q$-intersection pattern.*

Proof. We prove the lemma by constructing an appropriate hyperplane. We fix $n = n_0 + \cdots + n_{t-1}$, $\widehat{C}$, 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 $\widehat{C}$. 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 $\widehat{C}_{k^*}$ for the quasicube defined by $\widehat{C}$'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 is 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 the weaker bound is enough for now). 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), or both. The sum of two such elements cannot be zero (the sets are additively symmetric and disjoint, and no cancellation can occur).

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 equals 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 3.4, $\prod_{j<k}(n_{c_j}+1)$ 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 3.4 and 3.5, 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 $\widehat{C}$'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 consecutively 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 as above 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 3.10. The proof of Lemma 3.9 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 3.9 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 3.9.

Definition 3.11. A set family $\{S_n\subset\{0,1\}^n\}_{n\in\mathbb{N}}$ is *compact* if, for each $n\in\mathbb{N}$, S_n is expressible as a piecewise constant code whose cell representation $\widehat{S_n}$ admits a covering by polynomially many quasicubes.

Theorem 3.12. *If a set family $\{S_n\subset\{0,1\}^n\}_{n\in\mathbb{N}}$ is compact, 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 $\widehat{S_n}=\bigcup_{i=0}^{m-1}\widehat{C_i}$ for quasicubes $\widehat{C_i}\subset\prod_{i=0}^{t-1}\{0,\ldots,n_i\}$ (where m is polynomial in n), 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 $\widehat{C_i}$. The result follows immediately from Lemma 3.9. □

Theorem 3.12 is extremely powerful, and subsumes all hyperplane-covering constructions we're aware of. We immediately record the following corollary:

Corollary 3.13. *If a set family $\{S_n\subset\{0,1\}^n\}_{n\in\mathbb{N}}$ is such that each S_n is expressible as a piecewise constant code whose cell representation $\widehat{S_n}$ consists of only polynomially many filled cells, then $\{S_n\}_{n\in\mathbb{N}}$ is in* **H**.

Proof. As every cell is a quasicube (see Example 3.7), the hypothesis implies that $\{S_n\}_{n\in\mathbb{N}}$ is compact; the result thus follows directly from Theorem 3.12. □

We discuss specific consequences in the next subsection.

3.2 The complexity classes H and co-H

In this subsection, we undertake a thorough study of the complexity classes **H** and **co-H**. We establish a number of relations—both positive and negative—between these classes and standard classes in circuit complexity. We moreover study numerous specific examples. We use Theorem 3.12 as a primary tool in this process. Surprisingly, many natural boolean functions $f_n:\{0,1\}^n\to\{0,1\}$ have on-sets or off-sets which satisfy the hypothesis of Theorem 3.12. We begin with the following fundamental result:

Theorem 3.14. $\mathbf{\Sigma_2} \subset \mathbf{H}$ *and* $\mathbf{\Pi_2} \subset$ **co-H**.

Proof. As the two statements are equivalent, we only prove the first. Fixing an arbitrary family $\{S_n \subset \{0,1\}^n\}_{n\in\mathbb{N}}$ in $\mathbf{\Sigma_2}$, we have, by definition, that each $S_n \subset \{0,1\}^n$ is coverable by polynomially many subcubes; on the other hand, each S_n is of course piecewise constant with respect to the partition $n = \sum_{i=0}^{n-1} 1$ (see Example 3.8), in whose cell array these subcubes become quasicubes. The result thus follows directly from Theorem 3.12. □

More 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 $H : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{k-1} x_{c_i}$; this construction is correct provided that $q > n$.

Example 3.15. For even n, the function $f_n : (x_0, \ldots, x_{n-1}) \mapsto \bigvee_{i=0}^{n/2-1}(x_{2i} \wedge x_{2i+1})$ returns true if and only the bitwise AND of its argument's even-indexed and odd-indexed substrings contains a 1. Alternatively, f_n checks whether the respective subsets of $\{0, \ldots, \frac{n}{2}-1\}$ represented by these two substrings are non-disjoint. As the family $\left\{f_n^{-1}(1)\right\}_{n\in\mathbb{N}}$ is decided by a $\mathbf{\Sigma_2}$-circuit, we conclude from Theorem 3.14 that the $\{f_n\}_{n\in\mathbb{N}}$ is in $\mathbf{H}$. In fact, each f_n 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 3.12 (over and above Corollary 3.13) 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 $f_n^{-1}(1)$; these are correct so long as $q \geq 3$.

We now turn to the inclusions $\mathbf{\Pi_2} \overset{?}{\subset} \mathbf{H}$ and $\mathbf{\Sigma_2} \overset{?}{\subset}$ **co-H**. We study these inclusions through the following example, which exhibits many important properties.

Example 3.16. We continue our study of the function f_n of Example 3.15, and now consider its *off-set*. The function $f_n : (x_0, \ldots, x_{n-1}) \mapsto \bigvee_{i=0}^{n/2-1}(x_{2i} \wedge x_{2i+1})$ returns *false* if and only if the bitwise AND of its argument's even-indexed and odd-indexed substrings consists entirely of 0s. Alternatively, f_n returns false if and only if the subsets of $\{0, \ldots, \frac{n}{2}-1\}$ represented respectively by its argument's even-indexed and odd-indexed substrings are disjoint. The family $\{f_n^{-1}(0)\}_{n\in\mathbb{N}}$ is in $\mathbf{\Pi_2}$. As before, each off-set $f_n^{-1}(0)$ is piecewise constant with respect to $n = \sum_{i=0}^{n/2-1} 2$; its cell representation under this partition is exactly $\widehat{f_n^{-1}(0)} = \prod_{i=0}^{n/2-1}\{0,1\} \subset \prod_{i=0}^{n/2-1}\{0,1,2\}$.

The family $\{f_n^{-1}(0)\}_{n\in\mathbb{N}}$ of Example 3.16 above gives, among other things, a set family which is *not* compact in the sense of Definition 3.11.

Lemma 3.17. *The set family* $\{f_n^{-1}(0)\}_{n\in\mathbb{N}}$ *of Example 3.16 is not compact.*

Proof. We show that for any partition $n = \sum_{i=0}^{t-1} n_i$ of n with respect to which $f_n^{-1}(0)$ becomes piecewise constant, $f_n^{-1}(0)$'s resulting cell array representation requires exponentially many quasicubes to cover. To do this, we first argue that $f_n^{-1}(0)$ 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 $f_n^{-1}(0)$ is piecewise constant with respect to $\mathcal{F}$. Our hypothesis on $\mathcal{F}$ entails exactly that 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 $(0,1,1)$ at *some* appropriate element $(x_0, \ldots, x_{n-1}) \in f_n^{-1}(0)$, we conclude that $f_n^{-1}(0)$'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 positive. It follows that $f_n^{-1}(0)$ contains all of this cell's vectors, including at least one for which the components $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ attain the values $(1,1,0)$. This contradicts the definition of $f_n^{-1}(0)$. 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 $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ again attain the values $(0,1,1)$

at some appropriate element of $(x_0, \dots, x_{n-1}) \in f_n^{-1}(0)$. We conclude that $f_n^{-1}(0)$'s cell representation with respect to $\mathcal{F}$ contains a cell whose weight at F is at least two. This implies that the components $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ also attain the values $(1,1,0)$ at some element $(x_0, \dots, x_{n-1}) \in f_n^{-1}(0)$, which again contradicts the definition of $f_n^{-1}(0)$. 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 $\widehat{f_n^{-1}(0)}$ for $f_n^{-1}(0)$'s cell representation with respect to this partition. Because $\left|f_n^{-1}(0)\right| = 3^{n/2}$, it suffices to argue that, for each quasicube $\widehat{C}$ satisfying $\widehat{C} \subset \widehat{f_n^{-1}(0)}$, the pullback C of $\widehat{C}$ 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 $f_n^{-1}(0)$, at least one of these subsets' components must be bound for any quasicube $\widehat{C} \subset \widehat{f_n^{-1}(0)}$. Expanding the various cases, we see manually that each tuple $\{2i, 2i+1\}$ contributes a factor of at most 2 to the product expression defining C's size (for $i \in \{0, \dots, \frac{n}{2} - 1\}$). This completes the proof. □

I would like to thank Jason Long for suggesting the consideration of the function of Example 3.16.

In light of Lemma 3.17, is also interesting to ask whether the function family $\{f_n\}_{n\in\mathbb{N}}$ of Examples 3.15 and 3.16 belongs to **co-H**. Recent work of Diamond and Yehudayoff [DY22] settles exactly this question:

Theorem 3.18. *The function family $\{f_n\}_{n\in\mathbb{N}}$ of Examples 3.15 is not in* **co-H**.

Proof. By the main theorem of [DY22], any collection $\{H_i\}_{i=0}^{m-1}$ covering $f_n^{-1}(0)$ must satisfy $m = 2^{\Omega(n)}$. □

The following corollaries of Theorem 3.18 are immediate:

Corollary 3.19. $\mathbf{\Pi_2} \not\subset \mathbf{H}$ *and* $\mathbf{\Sigma_2} \not\subset$ **co-H**.

Corollary 3.20. $\mathbf{AC^0} \not\subset \mathbf{H}$ *and* $\mathbf{AC^0} \not\subset$ **co-H**.

Corollary 3.21. $\mathbf{H} \neq$ **co-H**.

Proof. Example 3.15's family $\{f_n\}_{n\in\mathbb{N}}$ satisfies $\left\{f_n^{-1}(1)\right\}_{n\in\mathbb{N}} \in \mathbf{H} - \mathbf{co\text{-}H}$ and $\left\{f_n^{-1}(0)\right\}_{n\in\mathbb{N}} \in \mathbf{co\text{-}H} - \mathbf{H}$. □

Even Theorem 3.18 does not furnish a function family $\{f_n\}_{n\in\mathbb{N}}$ whose on-sets and off-sets *simultaneously* fail to be efficiently coverable by hyperplanes. At the cost of adding a further layer to the circuit family deciding these functions, we easily produce such a family:

Theorem 3.22. $\mathbf{\Sigma_3} \not\subset \mathbf{H} \cup$ **co-H** *and* $\mathbf{\Pi_3} \not\subset \mathbf{H} \cup$ **co-H**.

Proof. The two statements are obviously equivalent, so we only prove the first. We again write $\{f_n\}_{n\in\mathbb{N}}$ for the function family of Examples 3.15 and 3.16; we define a new family $\{g_n\}_{n\in\mathbb{N}}$ in the following way. Fixing now n divisible by 4, we define $g_n : \{0,1\}^n \to \{0,1\}$ by setting, for each input $\mathbf{x} \in \{0,1\}^n$—with even-indexed and odd-indexed substrings $\mathbf{x}_0$ and $\mathbf{x}_1$, say—$g_n : \mathbf{x} \mapsto f_{n/2}(\mathbf{x}_0) \vee \overline{f_{n/2}(\mathbf{x}_1)}$. Clearly, $\left\{g_n^{-1}(1)\right\}_{n\in\mathbb{N}}$ is decided by a $\mathbf{\Sigma_3}$ circuit.

We now fix an *arbitrary* element $\mathbf{y}_0 \in f_{n/2}^{-1}(0)$. Given any hyperplane $H \subset \mathbb{F}_q^n$, the intersection of H with the $\frac{n}{2}$-dimensional affine subspace $Y_0 := \left\{(\mathbf{x}_0, \mathbf{x}_1) \in \{0,1\}^{n/2} \times \{0,1\}^{n/2} \;\middle|\; \mathbf{x}_0 = \mathbf{y}_0\right\}$ of $\mathbb{F}_q^n$ naturally induces an affine subspace of $\mathbb{F}_q^{n/2}$ of codimension at most one; we slightly abuse notation by writing $H \cap Y_0 \subset \mathbb{F}_q^{n/2}$ for this latter subspace. If moreover the condition $H \cap \{0,1\}^n \subset g_n^{-1}(1)$ holds, then, by definition of g_n, $H \cap Y_0 \cap \{0,1\}^{n/2} \subset f_{n/2}^{-1}(0)$ (and, in particular, $H \cap Y_0$ has the expected codimension of one, and so is a nondegenerate hyperplane).

We fix any collection of hyperplanes $\{H_i\}_{i=0}^{m-1}$ covering $g_n^{-1}(1)$. By the above discussion—and for $\mathbf{y}_0 \in \{0,1\}^{n/2}$ as above—the family of intersections $\{H_i \cap Y_0\}_{i=0}^{m-1}$ satisfies $H_i \cap Y_0 \cap \{0,1\}^{n/2} \subset f_{n/2}^{-1}(0)$ for each $i \in \{0, \dots, m-1\}$, and in fact gives a *covering* of $f_{n/2}^{-1}(0)$, containing *at most* m distinct hyperplanes. Applying Theorem 3.18, we conclude that $m \geq 2^{\Omega(n/2)}$. This concludes the argument that $\{g_n\}_{n\in\mathbb{N}} \notin \mathbf{H}$.

We now treat the family of off-sets $\{g_n^{-1}(0)\}_{n\in\mathbb{N}}$, using a similar argument. We fix an arbitrary element $\mathbf{y}_1 \in f_{n/2}^{-1}(1)$. Setting $Y_1 := \left\{(\mathbf{x}_0, \mathbf{x}_1) \in \{0,1\}^{n/2} \times \{0,1\}^{n/2} \;\middle|\; \mathbf{x}_1 = \mathbf{y}_1\right\}$, we see that each $H \subset \mathbb{F}_q^n$ such that $H \cap \{0,1\}^n \subset g_n^{-1}(0)$ satisfies $H \cap Y_1 \cap \{0,1\}^{n/2} \subset f_{n/2}^{-1}(0)$, by definition of g_n. It follows as before that for $\{H_i\}_{i=0}^{m-1}$ covering $g_n^{-1}(0)$, the family of intersections $\{H_i \cap Y_1\}_{i=0}^{m-1}$ gives a covering of $f_{n/2}^{-1}(0)$ of cardinality at most m, and hence that $m \geq 2^{\Omega(n/2)}$. We conclude that $\{g_n\}_{n\in\mathbb{N}} \notin$ **co-H**. This completes the proof. □

I would like to thank Amir Yehudayoff for suggesting the proof of Theorem 3.22.

Corollary 3.23. $\mathbf{AC}^0 \not\subset \mathbf{H} \cup \mathbf{co\text{-}H}$.

Another important class of examples is given by *symmetric* functions (see e.g. [Weg87, § 3.4]).

Theorem 3.24. *Any symmetric function $f_n : \{0,1\}^n \to \{0,1\}$'s on-set and off-set can be covered by n hyperplanes.*

Proof. By definition, any such f_n's on-set and off-set are both piecewise constant with respect to the partition $n = n$. Because the entire cell array of this partition has just $n+1$ cells, Corollary 3.13 implies that $n+1$ hyperplanes can cover these sets. In fact, it's easy to see that n hyperplanes suffice unless f_n is constant; ad-hoc constructions (using 2 or 0 hyperplanes, as the case may be) serve to settle this latter case. □

Concretely, to cover the on-set of a symmetric function $f_n : \{0,1\}^n \to \{0,1\}$, 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 the pullback of each filled cell $i \in \{0, \ldots, n\}$ (i.e., corresponding to each i for which $\{\mathbf{x} \in \{0,1\}^n \mid w(\mathbf{x}) = i\} \subset f_n^{-1}(1)$). This is correct so long as $q > n$.

Remark 3.25. The upper bound of Theorem 3.24 is the best possible, as is demonstrated by the classic lower bound of Alon and Füredi [AF93, Thm. 4] (concerning $f_n : (x_0, \ldots, x_{n-1}) \mapsto \bigvee_{i=0}^{n-1} x_i$).

Example 3.26. The *majority* function $f_n : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i \geq \left\lceil \frac{n}{2} \right\rceil$ is obviously symmetric; we conclude from Theorem 3.24 that $f_n^{-1}(0)$ and $f_n^{-1}(1)$ are each coverable by at most n hyperplanes (in fact, $\left\lceil \frac{n}{2} \right\rceil$ and $\left\lfloor \frac{n}{2} \right\rfloor$ suffice, respectively).

Corollary 3.27. $\mathbf{H} \not\subset \mathbf{AC}^0$ *and* $\mathbf{co\text{-}H} \not\subset \mathbf{AC}^0$.

Proof. We refer to Example 3.26 and the fact that majority is not in $\mathbf{AC}^0$ (see [Weg87, Cor. 3.32]). □

Example 3.28. For even n, the *Hamming-weight comparator* function $f_n : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} x_{2i} - x_{2i+1} \geq 0$ differs from majority by pre-composition with the affine bijection $(x_0, \ldots, x_{n-1}) \mapsto (x_0, 1 - x_1, \ldots, x_{n-2}, 1 - x_{n-1})$. Example 3.26 thus shows that $f_n^{-1}(1)$ can be covered using $\frac{n}{2}$ hyperplanes.

The following examples are extremely useful in practice.

Example 3.29. For even n, the function $f_n : (x_0, \ldots, x_{n-1}) \mapsto \bigwedge_{i=0}^{n/2-1} (x_{2i} \oplus \overline{x_{2i+1}})$ 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 $f_n : (x_0, \ldots, x_{n-1}) \mapsto \bigwedge_{i=0}^{n/2-1} (x_{2i} \oplus x_{2i+1})$. This latter function f_n's on-set $f_n^{-1}(1) \subset \{0,1\}^n$ 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 3.9, 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 $f_n^{-1}(1) = H \cap \{0,1\}^n$. Interestingly, this hyperplane returns exactly the integer whose *balanced ternary* representation is $(x_{2i} + x_{2i+1} - 1)_{i=0}^{n/2-1}$; it is correct so long as q exceeds the $\frac{n}{2}$-trit ternary "bias" $\sum_{i=0}^{n/2-1} 3^i = \frac{3^{n/2}-1}{2}$.

Example 3.30. Again for even n, we consider the integer comparison function $f_n : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} 2^i \cdot (x_{2i} - x_{2i+1}) > 0$ (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 $f_n : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} 2^i \cdot (x_{2i} + x_{2i+1}) \geq 2^{n/2}$ (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_n is such that $f_n^{-1}(1)$ is piecewise constant with respect to the partition $n = \sum_{i=0}^{n/2-1} 2$, and moreover that Theorem 3.12 applies to $\left\{f_n^{-1}(1)\right\}_{n \in \mathbb{N}}$ (in fact, the same conclusion holds for $\left\{f_n^{-1}(0)\right\}_{n \in \mathbb{N}}$, as can be shown by a similar treatment). Indeed, each f_n is evaluated by a certain variant of a standard comparison circuit, shown in Figure 1.

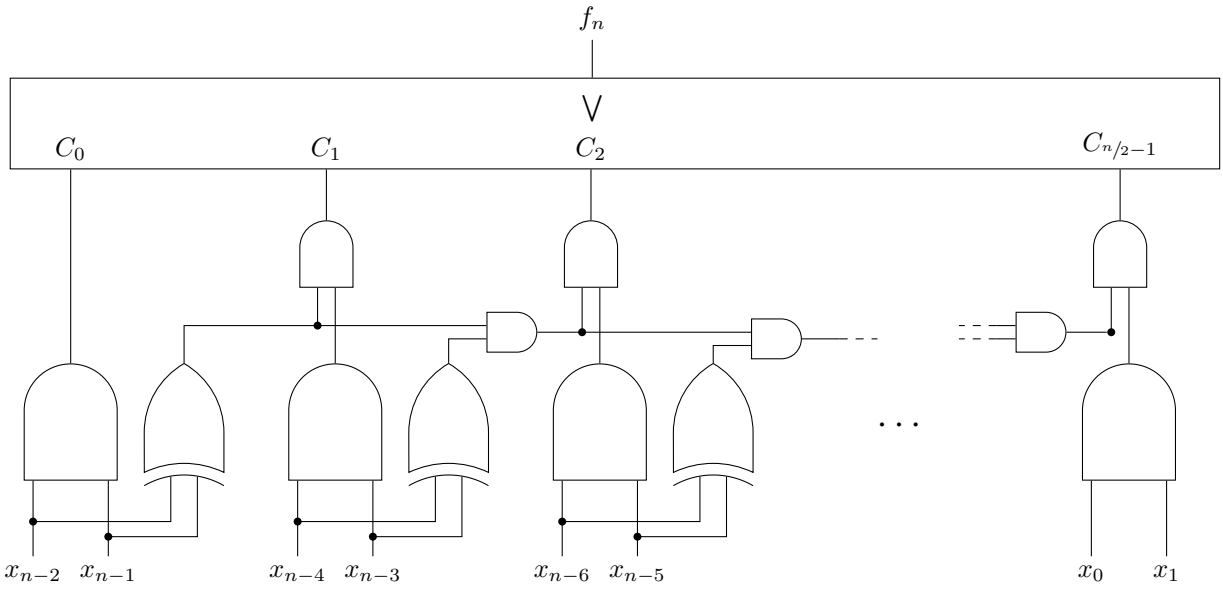

Figure 1: A well-known boolean circuit evaluating whether two unsigned integers' sum generates a carry.

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 $\widehat{C}_i \subset \prod_{i=0}^{n/2-1} \{0, 1, 2\}$ 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 $(v_0, \ldots, v_i) = (2, 1, \ldots, 1)$. Applying Theorem 3.12, we conclude that $f_n^{-1}(1)$ is coverable by $\frac{n}{2}$ hyperplanes.

Applying a version of the "special case" of the proof of Lemma 3.9, 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_n 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 3.31. 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). 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 $(2R - 2, 0, 2)$). Lemma 3.9 implies that S can be covered by exactly 4 hyperplanes; these are correct so long as the prime field order $q \geq (2R - 1) \cdot 4^2$. We conclude that the set family given by this construction belongs to **H** and that its family of complements belongs to **co-H**.

We finally present the following straightforward upper containment result.

Theorem 3.32. $\mathbf{H} \subset \mathbf{NC}^1$ *and* **co-H** $\subset \mathbf{NC}^1$.

Proof. Because $\mathbf{NC}^1$ is closed under complementation, it suffices to prove that $\mathbf{H} \subset \mathbf{NC}^1$. 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 $a_{i,0} \cdot x_0 + \cdots + a_{i,n-1} \cdot x_{n-1}$ evaluated by H_i, restricted to boolean inputs, is actually a *subset sum* (i.e., each $a_{i,j}$ 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 $a_{i,j}$ (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 can be carried out using a log-depth bounded fan-in circuit. The output of this circuit—namely, $a_{i,0} \cdot x_0 + \cdots + a_{i,n-1} \cdot x_{n-1}$—is an *integer* of bit-length $n + O(\log n)$; we must reduce this number modulo 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] (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]). Its output can obviously be checked for equality with v_i, the hyperplane's affine constant, in constant depth.

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. □

The construction of Theorem 3.32 is depicted in Figure 2 below.

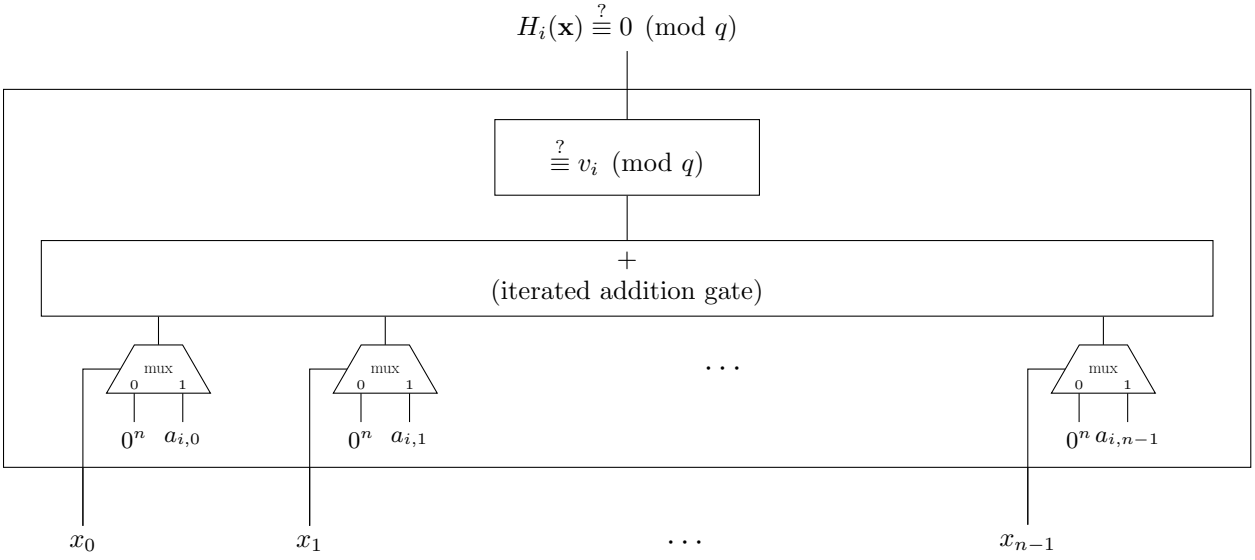

Figure 2: A log-depth, bounded fan-in boolean circuit evaluating an affine hyperplane.

We conclude this subsection with a few further remarks. It is tempting to formulate a converse to Theorem 3.12 of the previous subsection, especially in light of the evidence furnished jointly by Example 3.16, Lemma 3.17 and Theorem 3.18. As it turns out, the most naïve possible converse of Lemma 3.9 actually fails, in the sense that there exist intersection patterns which are *not* quasicubes with respect to *any* partition:

Example 3.33. For q any odd prime, in $\mathbb{F}_q^3$, $H : (x_0, x_1, x_2) \mapsto x_0 + x_1 - 2 \cdot x_2$ intersects $\{0,1\}^3$ exactly at $C = \{(0,0,0), (1,1,1)\}$. This latter set is not the pullback of a quasicube with respect to any partition of 3.

Nonetheless, it remains possible that an appropriately formulated *asymptotic* converse could hold. The difficulty of the result of Diamond and Yehudayoff [DY22] hints at the challenge which attaining any such general result would present; we leave this task as a direction for future work.

4 Commitment-Consistent 2PC

In this section, we describe a protocol for commitment-consistent secure two-party computation, efficient for function families in the classes **H** and **co-H**. We also give 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 $\left(\texttt{multiply}, \left(\langle\alpha_i\rangle_\nu, \langle\beta_i\rangle_\nu\right)_{i=0}^{m'-1}\right)$ from both parties P_ν, $\mathcal{F}_{\mathsf{PrivMult}}$ proceeds as follows:
  1: **for** $i \in \{0, \ldots, m'-1\}$ **do**
  2: set $\gamma_i := (\langle\alpha_i\rangle_0 + \langle\alpha_i\rangle_1) \cdot (\langle\beta_i\rangle_0 + \langle\beta_i\rangle_1) \pmod q$.
  3: randomly additively share $\gamma_i = \langle\gamma_i\rangle_0 + \langle\gamma_i\rangle_1 \pmod q$.

  $\mathcal{F}_{\mathsf{PrivMult}}$ sends the message $\left(\texttt{multiply}, \left(\langle\gamma_i\rangle_\nu\right)_{i=0}^{m'-1}\right)$ 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 of the following functionality from Lindell, Nof and Ranellucci [LNR18, Func. 4.2].

FUNCTIONALITY 4.4 ($\mathcal{F}_{\mathsf{CheckZero}}$—joint assessment of plaintext equality with zero)**.**
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, $\mathcal{F}_{\mathsf{CheckZero}}$ runs $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and outputs $(\texttt{key}, pk)$.
- Upon receiving $(\texttt{check}, A)$ from both parties, $\mathcal{F}_{\mathsf{CheckZero}}$ returns $\left(\texttt{check}, \mathsf{Dec}_{sk}(A) \stackrel{?}{=} 0\right)$ to both.

The result [LNR18, Prop. 7.2] yields a secure instantiation of $\mathcal{F}_{\mathsf{CheckZero}}$:

Lemma 4.5 (Lindell–Nof–Ranellucci)**.** *The protocol [LNR18, Prot. 7.1] securely computes* $\mathcal{F}_{\mathsf{CheckZero}}$ *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 our implementation of 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). Indeed, our randomized variant essentially returns *either* 0 or a uniformly random element $y \leftarrow \mathbb{F}_q$, depending on whether any among the parties' reconstructions $y_i := \langle y_i \rangle_0 + \langle y_i \rangle_1 \pmod q$ equals 0 or not.

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 $\mathcal{F}_{\mathsf{CheckZero}}$, 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:
  1: randomly sample $y \leftarrow \mathbb{F}_q$. ▷ to omit re-randomization, replace this assignment with $y := 1$.
  2: **for** $i \in \{0, \ldots, m-1\}$ **do**
  3: **for** $\nu \in \{0, 1\}$ **do**
  4: require $Y_{i,\nu} \stackrel{?}{=} \mathsf{Enc}_{pk}(\langle y_i \rangle_\nu ; s_{i,\nu})$; else send $(\texttt{multiply-abort})$ to both parties and abort.
  5: reconstruct $y_i := \langle y_i \rangle_0 + \langle y_i \rangle_1 \pmod q$.
  6: overwrite $y := y \cdot y_i \pmod q$.
  7: output $(\texttt{multiply}, y)$ 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. 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 $\mathcal{F}_{\mathsf{CheckZero}}$, and stores the response $(\texttt{key}, pk)$ from $\mathcal{F}_{\mathsf{CheckZero}}$.

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** $i \in \{0,\dots,m-1\}$ **do**
2: submit $(\texttt{prove}, Y_{i,\nu}; \langle y_i\rangle_\nu, s_{i,\nu})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$.
3: submit $(\texttt{verify}, Y_{i,1-\nu})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$.
4: **procedure** RecursiveMultiply $\left((\langle y_i\rangle_\nu, s_{i,\nu}, Y_{i,\nu}, Y_{i,1-\nu})_{i=0}^{m-1}\right)$
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 $(\langle y_{2i}\rangle_\nu, \langle y_{2i+1}\rangle_\nu)_{i=0}^{m'-1}$; assign to $\left(\langle y'_i\rangle_\nu\right)_{i=0}^{m'-1}$ the resulting output.
7: **for** $i \in \{0,\dots,m'-1\}$ **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 $\mathcal{F}_{\mathsf{CheckZero}}$.
17: **if** $m' > 1$ **then return** RecursiveMultiply $\left((\langle y'_i\rangle_\nu, s'_{i,\nu}, Y'_{i,\nu}, Y'_{i,1-\nu})_{i=0}^{m'-1}\right)$.
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 $\left((\langle y_i\rangle_\nu, s_{i,\nu}, Y_{i,\nu}, Y_{i,1-\nu})_{i=0}^{m-1}\right)$.
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* $\left(\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}, \mathcal{F}_{\mathsf{CheckZero}}\right)$*-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 whatever $\mathcal{A}$ outputs, and halts.

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

1. When $\mathcal{A}$ sends $(\texttt{init})$ to $\mathcal{F}_{\mathsf{CheckZero}}$, $\mathcal{S}$ 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 $\mathcal{F}_{\mathsf{CheckZero}}$.

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.

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 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 $\mathcal{F}_{\mathsf{CheckZero}}$:
        - 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** 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 $\left(\langle y'_0 \rangle_C, \langle y'_0 \rangle_{1-C}, s'_{0,C}, Y'_{0,C}, Y'_{0,1-C} \right) \leftarrow$ RecursiveSimulate $\left(\left(\langle y_i \rangle_C, 0, Y_{i,C}, Y_{i,1-C} \right)_{i=0}^{m-1} \right)$.
sample $a_{1-C} \leftarrow \mathbb{F}_q$ 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 $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, if the statement A_{1-C} is correct, 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 $\left(\langle y'_0 \rangle_C, \langle y'_0 \rangle_{1-C}, Y'_{0,C}, Y'_{0,1-C} \right) \leftarrow$ 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 $\left(\texttt{multiply}, (\langle y_i \rangle_C, s_{i,C})_{i=0}^{m-1} \right)$ to $\mathcal{F}_{\mathsf{IterMult}}$; receive $(\texttt{multiply}, y)$ 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.

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 $\mathsf{Real}_{\Pi,\mathcal{A},C}$.

Lemma 4.10. *If the underlying multiplication subprotocol Π_{PrivMult} is private in the sense of Definition 4.2, then the distributions $\{\mathsf{D}_0(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ and $\{\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ 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 $\mathsf{RecursiveSimulate}$ (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 $\{\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ and $\{\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ 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 $(a_0+a_1)\cdot\prod_{i=0}^{m-1}(\langle y_i\rangle_0 + \langle y_i\rangle_1)$ 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 $\{\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ and $\{\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ are computationally indistinguishable.*

Proof. We 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 as before 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 refining 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 $\mathsf{LR}_{pk,b}(\cdot,\cdot)$. $\mathcal{A}'$ simulates an interaction between $\mathcal{F}_{\mathsf{IterMult}}$, P_{1-C}, and the following modified version of $\mathcal{S}$. Instead of executing 1. above, $\mathcal{A}'$ internally simulates $\mathcal{F}_{\mathsf{CheckZero}}$ giving $\mathcal{A}$ $(\texttt{key}, pk)$, where pk is the experimenter's public key. Instead of executing 2. above, $\mathcal{A}'$, using P_{1-C}'s inputs, 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 $Y'_{i,1-C} \leftarrow \mathsf{LR}_{pk,b}\left(0, \langle y_{2i}\rangle_{1-C} \cdot (\langle y_{2i+1}\rangle_0 + \langle y_{2i+1}\rangle_1)\right)$ 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. We conclude that:

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

$$\geq \frac{1}{2} + \frac{1}{2 \cdot p(\lambda)},$$

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* $\{\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ *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 the various abort conditions respectively applied by $\mathcal{S}$ (in D_3) and by P_{1-C} and the various functionalities (in D_4). The equivalence of these conditions is essentially obvious, except except perhaps at $\mathcal{S}$'s check in line 16. In that line, $\mathcal{S}$ proceeds if and only if $\mathcal{A}$'s extracted plaintext $\langle y'_i \rangle_C$ and $\mathcal{S}$'s multiplication output $\langle y'_i \rangle_{1-C}$ 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 $\mathcal{F}_{\mathsf{CheckZero}}$'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 line 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_2). □

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 $\mathsf{RecursiveMultiply}$ 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 could eliminate entirely the $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$ proofs from lines 21 and 22 of Protocol 4.8; we have retained these above essentially to simplify the exposition.

Remark 4.17. We contrast Lemma 4.13 with the security argument [LNR18, Thm. B.1, Mult., 9. (b)]. There, to simulate $\mathcal{F}_{\mathsf{CheckZero}}$, $\mathcal{S}$ 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 $\mathcal{F}_{\mathsf{CheckZero}}$'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 $f_n : \{0,1\}^n \to \{0,1\}$—has been fixed. For notational simplicity, we treat only the case in which $\{f_n\}_{n \in \mathbb{N}}$ belongs to $\mathbf{H}$; an identical protocol in which the final output bit is flipped suffices to treat the case of $\mathbf{co\text{-}H}$.

In order to simplify the analysis of Protocol 4.18, we implement its "commitment" consistency using homomorphic *encryption*. We could just as well have used a homomorphic commitment scheme (compare Examples 2.18 and 2.19). Informally, we repurpose our encryption scheme as a perfectly binding commitment scheme. Our encryption-centric 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 (Joint evaluation of a collection of linear tests)**.**
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 $i \in \left\{0, \ldots, \frac{n}{2} - 1\right\}$ **do**
  randomly additively secret-share $x_{\nu,i} = \langle x_{\nu,i}\rangle_0 + \langle x_{\nu,i}\rangle_1$ in $\mathbb{F}_q$.
  sample $r_{\nu,i,\nu} \leftarrow \mathbb{F}_q$ and encrypt $A_{\nu,i} := \mathsf{Enc}_{pk}(x_{\nu,i}; r_{\nu,i,\nu})$.
  send $A_{\nu,i}$ and $\langle x_{\nu,i}\rangle_{1-\nu}$ to $P_{1-\nu}$, and submit $(\texttt{prove}, pk, A_{\nu,i}; x_{\nu,i}, r_{\nu,i,\nu})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}$.
  locally write $r_{\nu,i,1-\nu} := 0$, $A_{\nu,i,1-\nu} := \mathsf{Enc}_{pk}\left(\langle x_{\nu,i}\rangle_{1-\nu}; 0\right)$, and $A_{\nu,i,\nu} := A_{\nu,i} - A_{\nu,i,1-\nu}$.
  receive $A_{1-\nu,i}$ and $\langle x_{1-\nu,i}\rangle_\nu$ from $P_{1-\nu}$, and submit $(\texttt{verify}, pk, A_{1-\nu,i})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}$.
  locally write $r_{1-\nu,i,\nu} := 0$, $A_{1-\nu,i,\nu} := \mathsf{Enc}_{pk}\left(\langle x_{1-\nu,i}\rangle_\nu; 0\right)$, and $A_{1-\nu,i,1-\nu} := A_{1-\nu,i} - A_{1-\nu,i,\nu}$.
submit $\left(\texttt{prove}, pk_\nu, pk, A_\nu, \sum_{i=0}^{n/2-1} 2^i \cdot A_{\nu,i}; x_\nu, r_\nu, \sum_{i=0}^{n/2-1} 2^i \cdot r_{\nu,i,\nu}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}$.
submit $\left(\texttt{verify}, pk_{1-\nu}, pk, A_{1-\nu}, \sum_{i=0}^{n/2-1} 2^i \cdot A_{1-\nu,i}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}$.
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 $\left(\texttt{commit}, \left(\mathsf{Enc}_{pk}\left(\langle y_i\rangle_\nu; s_{i,\nu}\right)\right)_{i=0}^{m-1}\right)$ to $\mathcal{F}_{\mathsf{IterMult}}$.
upon receiving $\left(\texttt{commit}, \left(Y_{i,1-\nu}\right)_{i=0}^{m-1}\right)$ from $\mathcal{F}_{\mathsf{IterMult}}$, ensure that the ciphertexts match those of (2).
submit $\left(\texttt{multiply}, \left(\langle y_i\rangle_\nu, s_{i,\nu}\right)_{i=0}^{m-1}\right)$ to $\mathcal{F}_{\mathsf{IterMult}}$, and receive the output y.
output $y \stackrel{?}{=} 0$.

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 $|\Pr\left[P_{1-C}\left(\mathsf{D}_0(\mathbf{x}_0, \mathbf{x}_1, \lambda)\right) = v\right] - \left[P_{1-C}\left(\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\right) = v\right]|$ 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 $|\Pr\left[\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1\right] - \Pr\left[\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1\right]| \geq \frac{1}{p(\lambda)}$; 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 $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}$. 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 $A_{1-C,i} \leftarrow \mathsf{LR}_{pk,b}(0, x_{1-C,i})$ using an oracle call, for each $i \in \{0, \ldots, \frac{n}{2} - 1\}$. $\mathcal{A}'$ 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 $\{D_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ and $\{D_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ are computationally indistinguishable.*

Proof. We 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 $\{D_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ and $\{D_4(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ are identical.*

Proof. 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 11 and 12 of Protocol 4.18 essentially facilitate formal compliance with the interface of $\mathcal{F}_{\mathsf{IterMult}}$, and can be omitted in a real-life implementation of the protocol. 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 describe 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 (though certain parts could plausibly be parallelized to good effect).

Throughout our implementation, we take as $(\mathbb{G}, q, g)$ 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. 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 download and upload 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 3.30. 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 3.30), 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.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[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.

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

[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.

[CDN15] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, Cambridge, 2015.

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

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 494–503, New York, NY, USA, 2002. Association for Computing Machinery.

[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.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority – Or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security – ESORICS 2013*, pages 1–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[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.

[Dos05] Tomislav Došlić. Maximum product over partitions into distinct parts. *Journal of Integer Sequences*, 8, 2005.

[DY22] Benjamin E. Diamond and Amir Yehudayoff. Explicit exponential lower bounds for exact hyperplane covers. *Discrete Mathematics*, 345(11):113114, 2022.

[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.

[Gri93] Jerrold R. Griggs. On the distribution of the sums of residues. *Bulletin of the American Mathematical Society*, 28(2):329–333, 1993.

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

[IK00] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 294–304, 2000.

[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.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 365–391, Cham, 2018. Springer International Publishing.

[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.

[LO43] J. E. Littlewood and A. C. Offord. On the number of real roots of a random algebraic equation (III). *Sbornik: Mathematics*, 12(3):277–286, 1943.

[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.

[Raz92] A. A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, December 1992.

[RY20] Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, 2020.

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

[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.