Two-Round MPC without Round Collapsing Revisited – Towards Efficient Malicious Protocols*

Huijia Lin[1] and Tianren Liu[†2]

[1]University of Washington, Seattle, US
rachel@cs.washington.edu
[2]Peking University, Beijing, China
trl@pku.edu.cn

Abstract

Recent works have made exciting progress on the construction of round optimal, *two-round*, Multi-Party Computation (MPC) protocols. However, most proposals so far are still complex and inefficient.

In this work, we improve the simplicity and efficiency of two-round MPC in the setting with dishonest majority and malicious security. Our protocols make use of the Random Oracle (RO) and a generalization of the Oblivious Linear Evaluation (OLE) correlated randomness, called tensor OLE, over a finite field $\mathbb{F}$, and achieve the following:

- *MPC for Boolean Circuits:* Our two-round, maliciously secure MPC protocols for computing Boolean circuits, has overall (asymptotic) computational cost $O(S \cdot n^3 \cdot \log |\mathbb{F}|)$, where S is the size of the circuit computed, n the number of parties, and $\mathbb{F}$ a field of characteristic two. The protocols also make black-box calls to a Pseudo-Random Function (PRF).
- *MPC for Arithmetic Branching Programs (ABPs):* Our two-round, information theoretically and maliciously secure protocols for computing ABPs over a general field $\mathbb{F}$ has overall computational cost $O(S^{1.5} \cdot n^3 \cdot \log |\mathbb{F}|)$, where S is the size of ABP computed.

Both protocols achieve security levels inverse proportional to the size of the field $|\mathbb{F}|$.

Our construction is built upon the simple two-round MPC protocols of [Lin-Liu-Wee TCC'20], which are only semi-honest secure. Our main technical contribution lies in ensuring malicious security using simple and lightweight checks, which incur only a constant overhead over the complexity of the protocols by Lin, Liu, and Wee. In particular, in the case of computing Boolean circuits, our malicious MPC protocols have the same complexity (up to a constant overhead) as (insecurely) computing Yao's garbled circuits in a distributed fashion.

Finally, as an additional contribution, we show how to efficiently generate tensor OLE correlation in fields of characteristic two using OT.

*This article is an extended version of the paper to appear at CRYPTO 2022.

†The work was partially done when Liu was a postdoctoral researcher at University of Washington.

Contents

1 Introduction

Improving efficiency is a central theme in the design of cryptographic protocols. Two important aspects are *computational efficiency* and *round efficiency*. In the context of secure Multi-Party Computation (MPC) protocols, since the seminal works in the 80s [Yao82, GMW87, BGW88, CCD88], remarkable improvements have been made on both fronts.

- In the past decade, innovative design and implementation improvements have drastically reduced the computational cost of MPC, leading to efficient protocols more and more applicable to practical situations (e.g., the SPDZ protocols [DPSZ12] and its followup works).
- Another long line of researches on minimizing the round complexity of MPC recently culminated at the construction of *two-round* MPC protocols based on the (minimal) assumption of two-round Oblivious Transfer (OT) in the Common Reference String (CRS) model [BL18, GS18]. Two rounds are optimal even for achieving only semi-honest security and with trusted setups [FKN94, IK97].

However, so far, most two-round MPC protocols are complex and inefficient, especially those achieving malicious security (even in correlated randomness and/or trusted setup model). Encouraged by the efficiency improvement in the realm of many-round MPC in the past decade, this work strives to improve the simplicity and efficiency of two-round MPC in the malicious setting with dishonest majority. Existing techniques can be broadly classified as follows:

- *Round Collapsing:* Introduced by [GGHR14] and initially relying on strong primitives such as indistinguishability obfuscation (iO) or witness encryption [GP15, CGP15, DKR15, GLS15], the round collapsing approach was improved in [GS17, BL18, GS18, GIS18, BLPV18] to rely on just malicious 2-round OT. The complexity of this approach stems from applying the *garbling technique* (e.g., [Yao82, AIK04]) to the next step function of a many-round MPC protocol to collapse the number of rounds to two. The non-black-box use of the underlying MPC protocol hurts both asymptotic and concrete efficiency.
- *Using Generic Non-Interactive Zero Knowledge (NIZK):* This approach starts with designing two-round MPC that are semi-maliciously secure[1], and then transform them to maliciously secure ones by applying generic NIZK to detect deviation from the protocol specification. Two round semi-malicious protocols can be built either via the above round collapsing approach or using primitives supporting homomorphic computation, such as, multi-key fully homomorphic encryption [AJL+12, MW16, CM15, BP16, PS16, AJJM20] or homomorphic secret sharing [BGI16, BGI17, BGI+18, BGMM20]. Using NIZK to prove about the execution of the semi-malicious MPC protocols is inefficient and leads to non-black-box use of underlying assumptions.
- *MPC-in-the-Head [IKOS07, IPS08, IKSS21]:* Another generic method for strengthening weak security to strong security is the "MPC-in-the-head" transformations [IKOS07, IPS08]. The recent work by [IKSS21] showed how to perform such transformations in just two rounds. To obtain a two-round maliciously secure protocol, the transformation uses a two-round protocol with (enhanced) semi-honest security [GIS18, LLW20], to *emulate* the execution of another two-round protocol that is maliciously secure in the honest majority setting [IKP10, Pas12]. The overall complexity is the (multiplicative) compound complexity of both protocols.

[1] These are protocols secure against corrupted parties who follow the protocol specification but may choose its input and randomness arbitrarily.

We observe that existing designs of two-round malicious MPC all apply generic transformations – using garbling or NIZK or MPC – to some underlying MPC, which often leads to non-black-box constructions (with exceptions [GIS18, IKSS21]) and inefficient protocols. To improve the state-of-affairs, we consider protocols that use the Random Oracle (RO) and simple correlated randomness that can be efficiently generated in an offline phase, and aim for either information theoretic security or computational security with black-box use of simple cryptographic tools like Pseudo-Random Functions (PRFs). As seen in the literature, both the random oracle and the online-offline model are extremely successful settings for designing efficient cryptographic protocols.

Our Results: We present a ligitweight construction of 2-round malicious MPC protocols, using RO and an enhanced version of the Oblivious Linear Evaluation (OLE) correlation, called *tensor* OLE. The OLE correlation is the arithmetic generalization of the OT correlation over a finite field $\mathbb{F}$. It distributes to one party (a_1, b_1) and another (a_2, b_2) which are random elements in $\mathbb{F}$ subject to satisfying the equation $a_1 a_2 = b_1 + b_2$[2]. The tensor OLE correlation further generalizes the OLE correlation to higher dimension: For dimension $k_1 \times k_2$,

$$P_1 \text{ holds: } \mathbf{a}_1 \in \mathbb{F}^{k_1},\ \mathbf{B}_1 \in \mathbb{F}^{k_1 \times k_2}, \qquad P_2 \text{ holds: } \mathbf{a}_2 \in \mathbb{F}^{k_2},\ \mathbf{B}_2 \in \mathbb{F}^{k_1 \times k_2}$$
$$\text{where } \mathbf{a}_1, \mathbf{a}_2, \mathbf{B}_1, \mathbf{B}_2 \text{ are random, subject to } \mathbf{a}_1 \mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2 \ .$$

In our protocols, we will use pairwise tensor OLE correlation, with only small constant dimension, concretely 4×4 and 1×11. Such correlation can be generated efficiently in an offline phase using off-the-shelf OLE protocols [BCGI18, CDI+19]. We also show how to efficiently generate tensor OLE correlation for fields of characteristic two using OT, which in turn can be generated with good concrete efficiency [IKNP03, BCG+19]. We can further rely on pseudorandom correlation, which can be efficiently generated in using techniques described in [BCG+20].

Using tensor OLE correlation and RO, we obtain the following protocols, in the setting of static corruption and security with abort.

MPC FOR BOOLEAN CIRCUITS: Our first result is a construction of efficient two-round maliciously secure MPC protocols for general Boolean circuits. The protocols make use of RO and tensor OLE over a finite field $\mathbb{F}$ of characteristic two, as well as black-box calls to a PRF. (Note that we choose to not instantiate the PRF with RO, because the latter is used for a different purpose. To obtain standard-model protocols, we will employ the random oracle heuristic to replace RO with a real-life hash function. By separating PRF from RO, we reduce the use of heuristic.) When computing an n-ary circuit C, the overall computational costs of all parties is $O(|C| \cdot n^3 \cdot \log \mathbb{F})$. The security level of the protocol is inverse proportional to the field size $|\mathbb{F}|^{-1}$; thus, $\log |\mathbb{F}|$ can be viewed as the effective security parameter. More formally,

Theorem 1.1 (MPC for Boolean Circuits, informal)**.** *Let $\mathbb{F}$ be a finite field of characteristic two. Let C an n-ary Boolean circuit $C : \{0,1\}^{\ell_1} \times \cdots \times \{0,1\}^{\ell_n} \to \{0,1\}^{\ell}$. Assume the existence of a PRF F with security level $2^{-\kappa(\lambda)}$ where λ is the seed length.*

There exists a two-round MPC protocol Π that securely computes C, using RO and tensor OLE correlated randomness, making black-box calls to the PRF, and achieving

- *overall complexity $O(\Gamma \log \mathbb{F})$ for $\Gamma := n^3 \cdot |C|$, and*
- *ϵ-computational, malicious security with selective abort, against up to $n-1$ corruption, where the security level $\epsilon = \frac{O(\Gamma + n^2 \cdot Q_{\mathrm{RO}})}{|\mathbb{F}|} + \frac{O(n \cdot |C|)}{2^{\kappa(\log |\mathbb{F}|)}}$, and Q_{RO} is the number of random oracle queries that the adversary makes.*

[2]When the field is GF(2), OLE correlation coincides with the OT correlation.

More specifically, parties in our protocols communicate in total $O(\Gamma)$ elements in $\mathbb{F}$, perform in total $O(\Gamma)$ arithmetic operations in $\mathbb{F}$, use in total $O(\Gamma)$ pairs of tensor OLE correlated randomness of constant dimensions, and make in total $O(\Gamma)$ random oracle calls and $O(|C| \cdot n)$ PRF calls. In addition, we can enhance the protocol to have security with unanimous abort at the cost of increasing the complexity by an additive $\mathrm{poly}(n, \lambda)$ overhead.

Our construction follows the technique in [BMR90, DI05, LPSY15] developed in the context of constructing constant-round MPC. They showed that to securely computing a Boolean circuit C, it suffices to securely compute Yao's garbling of the circuit [Yao82]. Furthermore, the latter can be computed by a *degree three* polynomial f over a finite filed $\mathbb{F}$ of characteristic 2 – we call them the **distributed-Yao** polynomials. Therefore, designing 2-round protocols for general circuits boils down to designing 2-round protocols for computing the degree three distributed-Yao polynomials. This is indeed the approach taken by [LLW20]; however, they achieve only semi-honest security. In this work, we further achieve malicious security (see Lemma 1.3 below), and like [LLW20], our protocols incur only *constant overheads* – their overall asymptotically complexity is the same as the complexity of distributed-Yao polynomials.

We give slightly more detail on distributed-Yao polynomials. They compute Yao's garbled circuits in a special way: First, labels for a wire u has form $\boldsymbol{\ell}_{u,b} = s^{(1)}_{u,b} \| \ldots \| s^{(n)}_{u,b}$, where $s^{(i)}_{u,b} \in \mathbb{F}$ is a PRF key sampled by party P_i. Next, the garbled table for a gate g with input wire u, v and output wire o contains entries of the form $\boldsymbol{\ell}_{o,g(a,b)} \oplus (\bigoplus_i Y^{(i)}_{u,a}) \oplus (\bigoplus_i Y^{(i)}_{v,b})$, where $Y^{(i)}_{u,a}$ and $Y^{(i)}_{v,b}$ are pseudorandom one-time-pads generated via PRF using party P_i's keys $s^{(i)}_{u,a}$ and $s^{(i)}_{v,b}$ respectively (evaluating on different inputs). Hence, the output label is hidden as long as one of the PRF keys is hidden. These entries are additionally permuted using mask bits k_u, k_v which are additively shared among all parties. The important point made by [BMR90, DI05, LPSY15] is that the PRF evaluations can be done locally by each party, and given the PRF outputs as inputs to f, such a garbled circuit can be computed in just degree 3 in $\mathbb{F}$. Analyzing the distributed-Yao polynomial for a circuit C reveals that it contains $O(\Gamma) = O(|C| \cdot n^3)$ monomials over $\mathbb{F}$. In comparison, our MPC protocol implementing C has overall complexity $O(|C| \cdot n^3 \cdot \log |\mathbb{F}|)$ incurring only a constant-overhead over distributed-Yao.

MPC FOR ARITHMETIC BRANCHING PROGRAMS (ABPS): Using similar approach, we obtain efficient, two-round, MPC protocols for ABPs over field $\mathbb{F}$. Here, we compute instead the distributed version of the degree three randomized encoding of Applebaum, Ishai, and Kushilevitz (AIK) [AIK04] for ABPs. More precisely, for an ABP g, we shall compute the n-ary polynomial $f((\mathbf{x}_1, \mathbf{r}_1), \cdots, (\mathbf{x}_n, \mathbf{r}_n)) = \mathsf{AIK}_g((\mathbf{x}_1, \cdots \mathbf{x}_n);\ \Sigma_{i \in [n]} \mathbf{r}_i)$, where the randomness used for computing the AIK encoding is additively shared among all n parties. We refer to this polynomial the **distributed-AIK** polynomial. The complexity of the resulting MPC protocol is determined by the number of monomials in this polynomial, which is $O(\Gamma) = O(|g|^{1.5} n^2)$. However, different from the case for circuits, our two-round protocols now incur a factor $O(n)$ overhead. Constant-overhead can be retained by adding one more round. More formally,

Theorem 1.2 (MPC for ABPs, informal)**.** *Let $\mathbb{F}$ be a finite field. Let g be an n-ary arithmetic branching program over $\mathbb{F}$, $g : \mathbb{F}^{l_1} \times \cdots \times \mathbb{F}^{l_n} \to \mathbb{F}$. Denote by $|g|$ the size of the matrix $M_g(\cdot)$ describing g s.t. $\det(M_g(x)) = g(x)$ for any x.*

There exists a two-round MPC protocol Π that securely computes f, using RO and tensor OLE correlated randomness and achieving

- *overall complexity $O(\Gamma \cdot n \cdot \log |\mathbb{F}|)$ for $\Gamma = |g|^{1.5} n^2$ and*
- *ϵ-statistical, malicious security with abort, against up to $n - 1$ corruption where the statistical simu-*

lation error is $\epsilon = \frac{O(\Gamma \cdot n + n^2 Q_{\mathrm{RO}})}{|\mathbb{F}|}$ and Q_{RO} is the number of random oracle queries that the adversary makes.

Furthermore, there is a three-round protocol achieving the same as above, but with overall complexity $O(\Gamma \cdot \log |\mathbb{F}|)$.

More specifically, parties of the 3-round protocols communicate in total $O(\Gamma)$ elements in $\mathbb{F}$, perform in total $O(\Gamma)$ arithmetic operations in $\mathbb{F}$, and use in total $O(\Gamma)$ pairs of tensor OLE correlated randomness of constant dimensions.

MPC FOR DEGREE THREE POLYNOMIALS: The key that enables above theorems is our construction of, two-round, MPC protocols for computing degree *three* polynomials over an (arbitrary) sufficiently large finite field $\mathbb{F}$. Importantly, the protocol has **constant overhead** – when computing polynomials with Γ monomials over $\mathbb{F}$, our protocols have **overall complexity** $O(\Gamma \cdot \log |\mathbb{F}|)$. Furthermore, the protocol makes only black-box use to the underlying field $\mathbb{F}$.

In order to achieve constant overhead, we only require these protocols to achieve a weaker malicious security, called *security with output substitution*. Intuitively, the protocol ensures the usual privacy guarantee of honest parties inputs – that nothing about honest parties' inputs are revealed beyond the output $\mathbf{y} = f(\mathbf{x}_1, \cdots, \mathbf{x}_n)$. But the honest party may (unanimously) receive incorrect output – the adversary always learn $\mathbf{y}$, and can replace it with another output $\mathbf{y}'$ of its choice without honest parties noticing.

Lemma 1.3 (MPC for degree 3 polynomials, Informal)**.** *Let $\mathbb{F}$ be a finite field. Let f be an n-ary degree three polynomial over $\mathbb{F}$, $f : \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_n} \to \mathbb{F}^{\ell}$; denote by $|f|$ the number of monomials in f.*

There exists a two-round MPC protocol Π that securely computes f, using RO and tensor OLE correlated randomness and achieving the following:

- *overall complexity $O(|f|)$, and*
- *ϵ-statistical, malicious security with output substitution, against up to $n-1$ corruption, where the statistical simulation error is $\epsilon = \frac{O(|f| + n^2 Q_{\mathrm{RO}})}{|\mathbb{F}|}$ and Q_{RO} is the number of random oracle queries that the adversary makes.*

Our construction easily generalizes to computing constant degree polynomials with constant overhead, which might be of independent interests.

Using the above protocols to compute the distributed-Yao or distributed-AIK polynomials gives two-round protocols for circuits or ABPs respectively, but achieving only security with output substitution. We complement this by presenting a generic transformation that enhances security with output substitution to security with abort. Essentially, the transformation computes a related circuit (or ABP resp.) that computes not only the output, but also authenticates of the output using each party's private key (supplied as part of the input). Since security with output substitution ensures the privacy of honest parties' keys, the adversary can no longer substitute the output without being detected. This transformation incurs only a small additive overhead in the case of circuits, but a multiplicative overhead $O(n)$ in the case of ABPs. That's why our two-round ABP protocols do not achieve constant overhead over the complexity of distributed-AIK. We show a different transformation that uses one more round to recover constant overhead.

TENSOR OLE OVER $\mathsf{GF}(2^\lambda)$ FROM OT: As a final contribution, we construct an efficient 4-round protocol for generating the tensor OLE correlation over $\mathsf{GF}(2^\lambda)$ using OT.

Theorem 1.4 (Tensor OLE over $\mathsf{GF}(2^\lambda)$ from OT, Informal)**.** *There is a 4-round, statistically and maliciously secure two party computation protocol for sampling tensor OLE correlations, in the OT hybrid model.*

Our protocol is simple and efficient; in particular, it does not use any generic 2PC techniques such as garbling and zero-knowledge protocols. Thus, parties can run this efficient protocol to generate tensor OLE correlations in an offline stage using OT, which in turn can be generated with concrete efficiency [IKNP03, BCG+19].

Comparison with Prior Two-Round MPC As discussed before, prior 2-round MPC constructions can be categorized into three types depending on their main technique: 1) round collapsing, 2) using NIZK, and 3) MPC-in-the-head. Almost all protocols using round collapsing and all protocols using NIZK make non-black-box use of underlying cryptographic primitive (e.g., MPC and MKFHE etc.), and many of them have poor asymptotic efficiency (e.g., [BL18, GS18]). The only black-box constructions are [GIS18, IKSS21], which as we discuss below are less efficient than our protocols.

The construction of [GIS18] is in the OT correlation model and uses the round collapsing technique. To compute a Boolean circuit f, parties need to garble the next step functions of an information theoretically and maliciously secure MPC protocol Π for f making black-box calls to OT (e.g. [IPS08]). Let C_Π be the circuit induced by Π with depth d_Π and size $|C_\Pi|$. The overall communication complexity is at least $(d_\Pi|C_\Pi|n^2\lambda^2)$, where λ is the security parameter. Since d_Π is at least the depth d of f, and $|C_\Pi|$ at least the size of f. This leads to a dependency on $d \cdot |f|$, which is worse than our complexity.

The construction of [IKSS21] following the MPC-in-the-head approach uses a two-round protocol with (enhanced) semi-honest security such as [GIS18, LLW20], to emulate the execution of another two-round protocol that is maliciously secure in the honest majority setting [IKP10, Pas12]. Consider for instance, the complexity of [LLW20] is already $\Omega(|f|n^3\lambda)$ and a loose lower bound of the complexity of [IKP10] is $\Omega(Sn^5\lambda)$. The overall complexity is at least $\Omega(Sn^8\lambda^2)$.

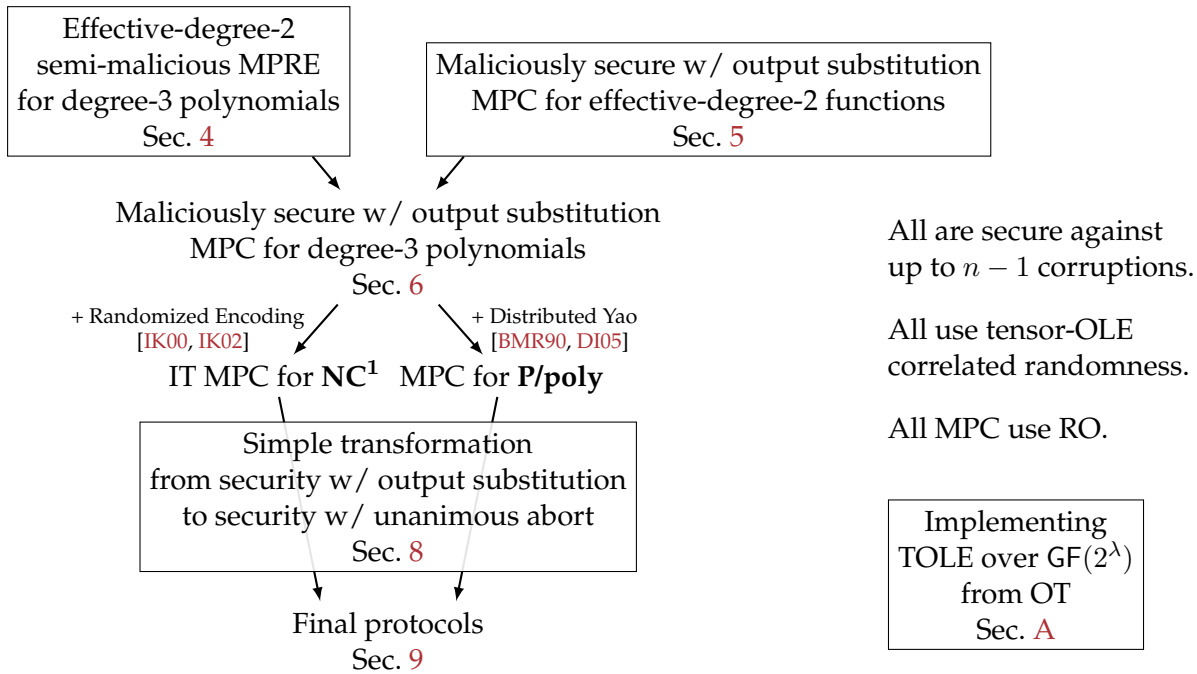

Figure 1: Our roadmap

2 Technical Overview

Our construction follows the overall structure of the semi-honest 2-round MPC protocols of [LLW20], which uses OLE correlated randomness. However, to make LLW maliciously secure, we face two challenges:

Challenge 1: Design *simple* and *efficient* checks to detect malicious behaviours.

To beat previous works, we avoid any generic transformation, such as, using generic NIZK, or even cryptographic operations. Our core protocol will be information-theoretic secure, and only use black-box operations over the field. Thus the detection of malicious behaviours can only rely on arithmetic methods.

Challenge 2: An adversary may lie about the *correlated randomness* it received. Such malicious behavior can hardly be caught even if we allow generic NIZK proof. This is because no party can write "using proper correlated randomness" as a NP-statement, and such statement naturally involves at least 2 parties who jointly hold the correlated randomness.

To deal with these challenges, we use tensor OLE correlated randomness instead of the scalar version, so that we have more room to play with arithmetic checks. We also use random oracle to generate challenges in the Fiat-Shamir style, so that our arithmetic proofs become non-interactive. We start with *security with output substitution*, and later transform to security with (unanimous) abort.

In the rest of the overview, we will walk through our constructions. We follow the successful paradigm of [IK00, IK02, ABT18, ABT19, LLW20]: reduce the task of securely computing a function f to the task of securely computing a simpler function $\hat{f}$. Such reduction is captured by a notion called MPRE. Sec. 2.1 revisits the definition of MPRE. Sec. 2.2 briefly presents our new MPRE, which reduces the task of computing general functions to the task of computing so-called "effective-degree-2" functions. Sec. 2.3 outlines 2-round malicious MPC protocols for computing effective-degree-2 functions. Composing them yields a 2-round MPC for general functions, but it only satisfies a weak notion of security. Sec. 2.4 outlines how to lift the security by a simple transformation. Our new constructions of MPRE and MPC are based on tensor OLE correlated randomness. In Sec. 2.5, we show how to generate such correlated randomness from OT. See Fig. 1 for a summary of the technical components and their sections in the technical body.

2.1 Multi-Party Randomized Encoding

The notion of multi-party randomized encoding (MPRE) is introduced by Applebaum, Brakerski and Tsabary [ABT18, ABT19]. In the correlated randomness model, for a n-party function f, an MPRE of f consists of n preprocessing functions $h_1, \ldots, h_n$, an encoding function $\hat{f}$, and a decoding function Dec, such that,

$$\mathsf{Decode}(\hat{f}(h_1(\mathbf{x}_1, \mathbf{r}_1, \mathbf{r}'_1), \cdots, h_n(\mathbf{x}_n, \mathbf{r}_n, \mathbf{r}'_n))) = f(\mathbf{x}_1, \cdots, \mathbf{x}_n) \ .$$

where the preprocessing function h_i is computed locally by party P_i on its input $\mathbf{x}_i$, randomness $\mathbf{r}_i$, and correlated randomness $\mathbf{r}'_i$. A semi-honest or malicious MPRE guarantees that to securely compute f, it suffices to securely compute $\hat{f}$ against semi-honest or malicious parties. That is,

the following canonical protocol computing f in the $\hat{f}$-hybrid world is semi-honestly (resp. maliciously) secure. [3]

The canonical protocol for MPRE Party P_i has input $\mathbf{x}_i$ and correlated randomness $\mathbf{r}'_i$, samples local randomness $\mathbf{r}_i$, computes $\hat{\mathbf{x}}_i = h_i(\mathbf{x}_i, \mathbf{r}_i, \mathbf{r}'_i)$ and feeds $\hat{\mathbf{x}}_i$ to the funtionality $F_{\hat{f}}$ computing $\hat{f}$, so that every party learns

$$\hat{y} := \hat{f}(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n)\,.$$

Then every party outputs $y = \mathsf{Dec}(\hat{y})$.

In other words, MPRE is a non-interactive reduction between MPC tasks. Thanks to the composition of MPC protocols, MPRE schemes naturally composes: Given an MPRE for f as described above, and another MPRE scheme for $\hat{f}$ with preprocessing functions $h'_1, \ldots, h'_n$, the encoding function $\hat{f}'$, their composition gives an MPRE for f with encoding function $\hat{f}'$ and preprocessing functions $h'_1 \circ h_1, \cdots, h'_n \circ h_n$. As such, as demonstrated in [ABT18, ABT19, LLW20], MPRE enables a modular approach for designing round-optimal MPC: To construct a round-optimal MPC protocol Π_f for computing f, the construction of [LLW20] proceeds in three steps:

- *Step 1: Degree 3 MPRE for circuits.* First, obtain a malicious MPRE for circuits, whose encoding function g has degree 3. It turns out that the classical degree 3 (centralized) randomized encoding, given by Yao's garbled circuits, is such a MPRE, where no local preprocessing is needed (i.e., h_i is the identity function). This has been implicitly observed and leveraged in many prior works, e.g., [BMR90, IK00, IK02, DI05, LLW20].
- *Step 2: Effective degree 2 MPRE for degree 3 Polynomials.* Then, design a MPRE of g whose encoding function $\hat{g}$ has degree 2. In this step, the preprocessing functions are non-trivial and such MPRE is said to have effective degree 2.
- *Step 3: 2-round MPC for degree 2 polynomials.* Finally, design a round-optimal MPC protocol $\Pi_{\hat{g}}$ for computing $\hat{g}$.

Composing the MPRE schemes from the first two steps gives an MPRE for circuit f with encoding function $\hat{g}$. The desired protocol Π_f is then obtained by instantiating the $\hat{g}$-oracle in the canonical protocol with $\Pi_{\hat{g}}$. Note that Π_f has the same communication complexity and round complexity as $\Pi_{\hat{g}}$. The contribution of [LLW20] lies in giving efficient instantiation of Step 2 and 3 in the OLE correlated randomness model over a field $\mathbb{F}$ of characteristic 2, and their final protocol Π_f has complexity $O(|f|n^3 \log |\mathbb{F}|)$. The main drawback is that their protocols are only semi-honest secure.

Semi-Malicious MPRE Our construction improves upon [LLW20] to achieve malicious security. Towards this, we introduce *semi-malicious* MPRE. As the name suggested, a MPRE of f is semi-maliciously secure if its canonical protocol is semi-maliciously secure, i.e., against adversaries who may choose arbitrary local randomness $\mathbf{r}_i$ and correlated randomness $\mathbf{r}'_i$ of corrupted parties, but computes the preprocessing functions correctly. Equivalently, semi-malicious MPRE means the following protocol is maliciously secure in the $\hat{f} \circ h$-hybrid world.

[3] An equivalent definition of semi-honest MPRE can be found in [ABT18], in which it is just called "MPRE". In [ABT19], malicious MPRE is called "non-interactive reduction" and the canonical protocol of a MPRE is called "$\hat{f}$-oracle-aided protocol". Both [ABT18] and [ABT19] consider the honest majority setting, so they only require the canonical protocol to be secure against a bounded number of corruptions.

The canonical protocol for semi-malicious MPRE Party P_i has input $\mathbf{x}_i$, samples local randomness $\mathbf{r}_i$, and receives $\hat{\mathbf{r}}_i$, where $(\hat{\mathbf{r}}_1, \ldots, \hat{\mathbf{r}}_n)$ is the correlated randomness. Every P_i feeds $(\mathbf{x}_i, \mathbf{r}_i, \hat{\mathbf{r}}_i)$ to an oracle computing $\hat{f} \circ h$, so that every party learns

$$\hat{y} = (\hat{f} \circ h)(\mathbf{x}_1, \mathbf{r}_1, \hat{\mathbf{r}}_1, \ldots, \mathbf{x}_n, \mathbf{r}_n, \hat{\mathbf{r}}_n) := \hat{f}(h_1(\mathbf{x}_1, \mathbf{r}_1, \hat{\mathbf{r}}_1), \ldots, h_n(\mathbf{x}_n, \mathbf{r}_n, \hat{\mathbf{r}}_n)) .$$

Then every party outputs $y = \mathsf{Dec}(\hat{y})$.

Now, in order to construct 2-round malicious MPC protocol for general circuits, we modify the second and third steps above to the following

- *Step 2:* **Semi-malicious** *Effective degree 2 MPRE for degree 3 Polynomials.* Design a *semi-malicious* MPRE for any degree-3 function f, whose encoding function $\hat{f}$ has degree 2;
- *Step 3: 2-round MPC for* **effective** *degree 2 polynomials.* Construct 2-round *malicious* MPC for computing $\hat{f} \circ h$, which is an effective-degree-2 function.

Composing the above two steps gives a round-optimal maliciously secure MPC protocol for degree 3 functions (Lemma 1.3); using it to compute the degree 3 maliciously secure MPRE for circuit f from Step 1 gives a round-optimal maliciously secure MPC protocol for f[4] .

Next, we outline our instantiation for Step 2 in Sec. 2.2 and that for Step 3 in Sec. 2.3. The entire roadmap is illustrated in Fig. 1.

2.2 Semi-Malicious Effective-Degree-2 MPRE

This section will outline the construction of semi-malicious effective-degree-2 MPRE for any degree-3 function. We start by "canonicalizing" the degree-3 function. A degree-3 polynomial can always be written as the sum of monomials $\sum_t c_t x_{t,1} x_{t,2} x_{t,3}$, where c_t is the constant coefficient of the t-th monomial. Let the party holding $x_{t,j}$ also sample random $z_{t,j}$, then

$$\Big(x_{t,1}x_{t,2}x_{t,3} + z_{t,1} + z_{t,2} + z_{z,3} \text{ for each } t, \sum\nolimits_t c_t(z_{t,1} + z_{t,2} + z_{z,3}) \Big), \tag{1}$$

is (the encoding function of) a malicious MPRE for $\sum_t c_t x_{t,1} x_{t,2} x_{t,3}$, as shown in [BGI+18, GIS18, LLW20]. So it suffices to construct semi-malicious effective-degree-2 MPRE for (1). Here (1) is in what we call *canonical form*: Every coordinate of $\hat{f}$ either linear, or looks like $x_1x_2x_3 + z_1 + z_2 + z_3$.

As we are constructing *semi-malicious* MPRE, it is fine to construct MPRE for each coordinate of (1) separately then simply concatenate them together. That is, it suffices to consider the complete 3-party functionality

$$\mathsf{3MultPlus}((x_1, z_1), (x_2, z_2), (x_3, z_3)) := x_1x_2x_3 + z_1 + z_2 + z_3 .$$

3MultPlus has a semi-honest effective-degree-2 MPRE (Fig. 2), which will be recalled in Sec. 4.1. For the overview, what matters is that

- This MPRE uses scalar OLE correlated randomness. That is, party P_1 receives $a_1, b_1 \in \mathbb{F}$, party P_2 receives a_2, b_2 such that a_1, a_2, b_1, b_2 are random subject to $a_1a_2 = b_1 + b_2$.

[4]The reader may have observed another approach: Combining Step 1 and 2 yields a semi-malicious MPRE for f, whose encoding function has degree 2; then compose it with Step 3 yields the desired protocol. The disadvantage of this approach is that the communication complexity of the final MPC will depends on the complexity of the preprocessing functions of Step 1.

	Party P_1	Party P_2	Party P_3
Input:	x_1, z_1	x_2, z_2	x_3, z_3
Local Randomness:	$a_{4,1}, a_{5,1}$	$a_{4,2}, a_{5,2}$	a_3
Correlated Randomness:	a_1, b_1	a_2, b_2	
Output:	$\begin{bmatrix} x_1 - a_1 & \binom{a_3 x_1 + a_1 x_3}{-a_1 a_3 - a_4} & \binom{b_1 x_3 + b_2 x_3 + a_5 x_1 - a_1 a_5 +}{a_4 x_2 - a_2 a_4 + z_1 + z_2 + z_3} \\ -1 & x_3 - a_3 & a_2 x_3 - a_5 \\ & -1 & x_2 - a_2 \end{bmatrix}$ where $a_4 := a_{4,1} + a_{4,3}$ and $a_5 := a_{5,2} + a_{5,3}$.		

Figure 2: Semi-honest MPRE for 3MultPlus [LLW20]

- This MPRE has perfect semi-honest security.

Our roadmap requires a semi-malicious effective-degree-2 MPRE for 3MultPlus. Semi-malicious security means the corrupted parties may arbitrarily choose their local randomness or modify the correlated randomness they received.

In standard model, semi-malicious security is implied by perfect semi-honest security, because conditional on any choice of corrupted parties' local randomness, the adversary has no advantage. But in the correlated randomness model, the semi-malicious adversary may lie about correlated randomness.

- For example, say P_1, P_2 are corrupted. If they feed a_1, b_1, a_2, b_2 as correlated randomness such that $a_1 a_2 = b_1 + b_2$, then the privacy still follows from the perfect semi-honest security. But if P_1, P_2 choose a_1, b_1, a_2, b_2 such that $a_1 a_2 \neq b_1 + b_2$, the privacy is lost (as we will show in Sec. 4.2).

- For another example, say P_1 is corrupted. If corrupted P_1 does not modify the portion of correlated randomness (a_1, b_2) it received, then the privacy still follows from the perfect semi-honest security. But if P_1 lies about (a_1, b_2), then with overwhelming probability $a_1 a_2 \neq b_1 + b_2$, and the privacy is lost (as we will show in Sec. 4.2).

In either case, if honest P_3 want to protect its privacy, it needs (and suffices) to ensure that $a_1 a_2 \neq b_1 + b_2$.

Our solution against this privacy threat is called *conditional disclosure of secrets (CDS) encoding*. Let party P_3 locally sample a random mask s. CDS encoding is a sub-module (of the final MPRE) that reveals s if and only if $a_1 a_2 \neq b_1 + b_2$. Then a candidate MPRE consists of two parts:

i) the semi-honest MPRE for 2MultPlus one-time padded by s;

ii) CDS encoding, which reveals s if and only if $a_1 a_2 = b_1 + b_2$.

But the CDS encoding resolves P_3's privacy concern at a cost: The adversary will learn a linear function in (a_1, b_1, a_2, b_2) if P_3 is corrupted. To overcome it, we have to replace the scalar OLE correlated randomness in the semi-honest MPRE by *tensor OLE correlated randomness*. That is, party P_1 receives vector $\mathbf{a}_1$, matrix $\mathbf{B}_1$; party P_2 receives vector $\mathbf{a}_2$, matrix $\mathbf{B}_2$; such that $\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}_2, \mathbf{B}_2$ are random subject to $\mathbf{a}_1 \mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$. We abuse the notation and let a_1, b_1, a_2, b_2 denote the first coordinate of $\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}_2, \mathbf{B}_2$ respectively. Then a_1, b_1, a_2, b_2 are random subject to $a_1 a_2 = b_1 + b_2$, i.e., their distribution is scalar OLE correlated randomness.

The next candidate MPRE is made up of:

	Party P_1	Party P_2	Party P_3
Input:	x_1, z_1	x_2, z_2	x_3, z_3
Local Randomness:	$a_{4,1}, a_{5,1}$ s_1	$a_{4,2}, a_{5,2}$ s_2	a_3 s_3
Correlated Randomness:	$\mathbf{a}_1, \mathbf{B}_1$	$\mathbf{a}_2, \mathbf{B}_2$	
Output includes:	[LLW20] MPRE for 3MultPlus $+ s_1 + s_2 + s_3$		
	CDS Encoding reveals s_1 if $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$	CDS Encoding reveals s_2 if $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$	CDS Encoding reveals s_3 if $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$

Figure 3: Semi-Malicious MPRE for 3MultPlus

i) the semi-honest MPRE for 2MultPlus one-time padded by s;

ii) CDS encoding, which reveals s if and only if $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$.

Additionally, CDS encoding will let the adversary learn a linear leakage function in $(\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}_2, \mathbf{B}_2)$ if P_3 is corrupted. But due to our careful design of CDS encoding, the leakage is one-time padded by the remaining coordinates of $\mathbf{a}_1, \mathbf{a}_2, \mathbf{B}_1, \mathbf{B}_2$, so that no information about a_1, b_1, a_2, b_2 is revealed.

So far we focus on the security concern of P_3. Party P_1, P_2 have similar concern. So in the actual semi-malicious MPRE for 3MultPlus (shown in Fig. 3), every party P_i locally sample random mask s_i, The final MPRE consists of:

i) the semi-honest MPRE for 2MultPlus one-time padded by $s_1 + s_2 + s_3$;

ii) (for each $i \in \{1, 2, 3\}$) CDS encoding that reveals s_i iff $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$.

Of course, the three instances of CDS encoding are carefully designed, so that their leakages jointly reveal no information about a_1, b_1, a_2, b_2.

In the rest of the section, we explain how CDS encoding works. W.l.o.g., we assume the secret mask is sampled by P_3.

By our discussion so far, it seems that P_3 has to sample a random matrix s_3 in order to one-time pad the [LLW20] MPRE. But as we will discover in the technical body, it is sufficient to one-time pad only the top right coordinate of the [LLW20] MPRE. So s_3 is a random scalar sampled by P_3.

We start with a simpler task, P_3 only want to verify if P_1, P_2 use legit tensor OLE correlation. Here "legit" means in the support of the distribution. A simple encoding is to let P_3 additionally sample random vectors $\mathbf{q}_1, \mathbf{q}_2$, and the encoding outputs

$$p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle, \quad p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle, \quad p_3 = \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1\mathbf{q}_2^\intercal \rangle \,. \tag{2}$$

Then P_3 can check whether $p_1p_2 = p_3$. If $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$, then $p_1p_2 = p_3$ always holds. Otherwise, $p_1p_2 \neq p_3$ with overwhelming probability. Note that the encoding in (2) is of effective degree 2, because P_3 can locally compute $\mathbf{q}_1\mathbf{q}_2^\intercal$.

In the CDS encoding, party P_3 additionally samples random r_1, r_2. The CDS encoding outputs

$$p_1, \quad p_2, \quad p_3, \quad \text{and} \quad \mathbf{c} := \begin{bmatrix} 1 & p_2 \\ p_1 & p_3 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} + \begin{bmatrix} 0 \\ s_3 \end{bmatrix}.$$

If $\mathbf{a}_1\mathbf{a}_2^\intercal \neq \mathbf{B}_1 + \mathbf{B}_2^\intercal$ then $\left[\begin{smallmatrix} 1 & p_2 \\ p_1 & p_3 \end{smallmatrix}\right]$ has full-rank with overwhelming probability, and $\mathbf{c}$ is one-time padded by (r_1, r_2), thus no information about s is revealed. Otherwise when $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$, it is easy to verify that $\langle(-p_1, 1), \mathbf{c}\rangle = s$. Note that CDS encoding is of effective degree 2, because it is a linear function in $\mathbf{a}_1, \mathbf{a}_2, \mathbf{B}_1, \mathbf{B}_2$, whose coefficient can be locally computed by P_3.

If P_3 is corrupted, the adversary chooses $\mathbf{q}_1, \mathbf{q}_2$ and learns p_1, p_2, p_3 (defined by (2)) as leakage. By enforcing some constraints on $\mathbf{q}_1, \mathbf{q}_2$ (will be discussed in the main body), the leakage will not reveal any information about a_1, b_1, a_2, b_2.

2.3 MPC for Effective-Degree-2 Functions

Given an effective-degree-2 function $g = \hat{f} \circ h$

$$g(\mathbf{x}_1, \ldots, \mathbf{x}_n) := \hat{f}(h_1(\mathbf{x}_1), \ldots, h_n(\mathbf{x}_n)),$$

we can assume w.l.o.g. that each coordinate of $\hat{f}$ has the canonical form[5]

$$\hat{f}_t(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n) = \mathsf{2MultPlus}(\underbrace{(x_1, z_1)}_{\text{owned by } P_{i_t}}, \underbrace{(x_2, z_2)}_{\text{owned by } P_{j_t}}) = x_1x_2 + z_1 + z_2$$

where x_1, z_1 are two coordinates of $\hat{\mathbf{x}}_{i_t}$ and x_2, z_2 are two coordinates of $\hat{\mathbf{x}}_{j_t}$.

As presented by [LLW20], there is a 2-round semi-honest MPC protocol for 2MultPlus that uses scalar OLE correlated randomness (Fig. 4). Via parallel repetition, it implies 2-round semi-honest MPC for any effective-degree-2 functions that uses scalar OLE correlated randomness.

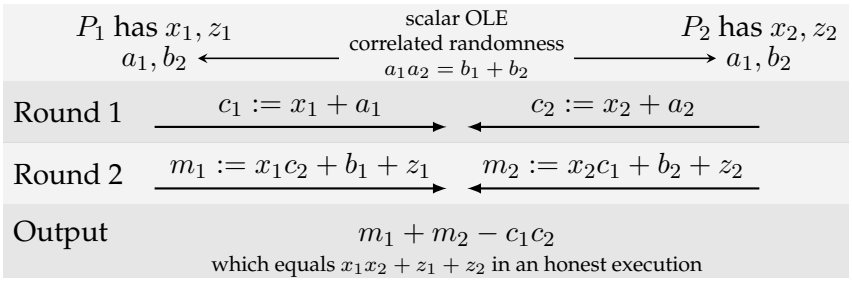

Figure 4: 2-round semi-honest MPC for 2MultPlus

Towards malicious security, the starting point is the observation that the protocol for 2MultPlus in Fig. 4 is maliciously secure *with output substitution*. Security with output substitution means the adversary, after learning the output y, may adaptively choose y' so that all honest parties (unanimously) output y'.

A natural idea is to compute all $\hat{f}_t$ using parallel sessions of the protocol in Fig. 4. Each session computes one coordinate. But parallelization does not meet the security requirement, because of the following two issues:

[5]Sec. 2.2 outlines how to canonicalize $\hat{f}$. Formally, canonical form allows some coordinates to be linear instead of 2MultPlus. The linear coordinates are easier to handle. We ignore them in the overview.

Consistency. Say two coordinates of $\hat{f}$ equal $x_1x_2 + z_1 + z_2$ and $x_1x_3 + z_1' + z_3$ respectively, where party P_1 owns x_1, z_1, z_1', party P_2 owns x_2, z_2, party P_3 owns x_3, z_3. We have to check whether P_1 feeds the *same* x_1 to the two corresponding sessions.

Well-Formedness. We have to check whether P_i computes $\hat{\mathbf{x}}_i = h_i(\mathbf{x}_i)$ correctly. Note that we can also assume the preprocessing functions h_i is a small arithmetic circuit that only contains multiplication gates, because such property is satisfied by our MPRE.

To resolve the well formedness issue, we introduce a commit-and-prove-linear scheme which uses vector OLE correlated randomness. It enables the sender to commit to a vector $\mathbf{x}$, then later prove $L(\mathbf{x}) = y$ for any linear function L.

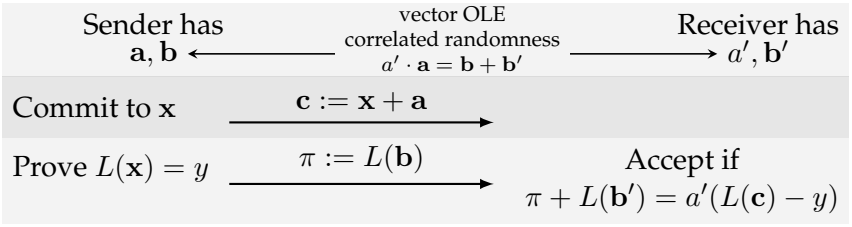

Figure 5: commit-and-prove-linear

As shown in Fig. 5, the scheme uses vector OLE correlated randomness between the sender and the receiver. That is, the sender receives random vectors $\mathbf{a}, \mathbf{b}$, the receiver recovers random $a', \mathbf{b}'$ subject to $a'\mathbf{a} = \mathbf{b} + \mathbf{b}'$. To commit to a vector $\mathbf{x}$, the sender simply sends $\mathbf{c} := \mathbf{x} + \mathbf{a}$ to the receiver. Later, for any linear function L, the sender can prove $L(\mathbf{x}) = y$ by sending $\pi := L(\mathbf{b})$ to the receiver; and the receiver accepts the proof iff $\pi + L(\mathbf{b}') = a'(L(\mathbf{c}) - y)$. Such scheme has

Completeness An honest proof $\pi := L(\mathbf{b})$ will always be accepted because

$$\pi + L(\mathbf{b}') = L(\mathbf{b}) + L(\mathbf{b}') = a' \cdot L(\mathbf{a}) = a'(L(\mathbf{c}) - L(\mathbf{x})) = a'(L(\mathbf{c}) - y)\,.$$

Statistical Soundness If $L(\mathbf{x}) \neq y$, any proof will be reject with overwhelming probability due to the randomness of a'.

Zero-knowledge No information about $\mathbf{x}$, other than $L(\mathbf{x})$, will be revealed to receiver, since the receiver can predict π from $\mathbf{c}, L, y$.

Reusability Given a commitment $\mathbf{c}$, the sender can prove multiple adaptively chosen linear statements $L_1(\mathbf{x}) = y_1, \ldots, L_t(\mathbf{x}) = y_t$ about the underlying message $\mathbf{x}$. The statistical soundness error is $O(t/|\mathbb{F}|)$. No information about $\mathbf{x}$ other than $L_1(\mathbf{x}), \ldots, L_t(\mathbf{x})$ will be revealed.

Then, inspired by a linear PCP scheme from [BCI+13], the sender can generate zero-knowledge proofs of more complex arithmetic statements. A particular interesting statement for ours is multiplication. Say the sender will commit to message $\mathbf{x} = (x_1, x_2, x_3, \ldots)$, and then prove $x_1x_2 = x_3$ to the receiver. We design **ProveProd** sub-protocol for such demand:

1. In order to prove $x_1x_2 = x_3$, the sender samples random g_1, g_2 and extends its message into

   $$\mathbf{x} = (x_1, g_1, x_2, g_2, x_3, x_1g_2, x_2g_1, g_1g_2, \ldots)\,.$$

   The dimension of the correlated randomness should be extended correspondingly. The sender commits to the extended $\mathbf{x}$ instead.

2. Random challenges q_1, q_2 are sampled.

3. The sender proves to the receiver that

$$\langle (q_1, 1), (\underset{\substack{\uparrow \\ (x_1, g_1)}}{\mathbf{x}[1]}, \mathbf{x}[2]) \rangle = p_1, \quad \langle (q_2, 1), (\underset{\substack{\uparrow \\ (x_2, g_2)}}{\mathbf{x}[3]}, \mathbf{x}[4]) \rangle = p_2,$$
$$\langle (q_1 q_2, q_1, q_2, 1), (\mathbf{x}[5], \mathbf{x}[6], \underset{\substack{\uparrow \\ (x_3, x_1 g_2, x_2 g_1, g_1 g_2)}}{\mathbf{x}[7]}, \mathbf{x}[8]) \rangle = p_3 \,.$$

   The receiver accepts if $p_1 p_2 = p_3$.

Similar to the discussion in our CDS encoding, if $\overset{x_1}{\downarrow}\mathbf{x}[1] \cdot \overset{x_2}{\downarrow}\mathbf{x}[3] \neq \overset{x_3}{\downarrow}\mathbf{x}[5]$, then $p_1 p_2 \neq p_3$ with overwhelming probability due to the randomness of $q_1 q_2$. No information about x_1, x_2, x_3 are leaked if the sender is proving a true statement, as the leakage p_1, p_2 are one-time padded by g_1, g_2, and p_3 is determined by $p_3 = p_1 p_2$.

If the random challenges q_1, q_2 are sampled by the receiver, the proof will have $1/|\mathbb{F}|$ soundness error. To make the proof non-interactive, the challenges can be sampled by the random oracle, so that the soundness error becomes the number of adversary's query to random oracle divided by $|\mathbb{F}|$.

So far we can prove multiplication relation for 3 values behind a commitment. This will resolve the concern of well-formedness. To resolve the concern of consistency, we need to prove that values behind different commitments are the same. That is, say the sender will commit to messages $\mathbf{x}_1 = (x_1, \dots)$, $\mathbf{x}_2 = (x_2, \dots)$, and want to convince the receiver that $x_1 = x_2$. We design **ProveSame** sub-protocol for such proof:

1. In order to prove $x_1 = x_2$, the sender samples random g and extends its message into

$$\mathbf{x}_1 = (x_1, g, \dots), \quad \mathbf{x}_2 = (x_2, g, \dots) \,.$$

   The sender commits to the extended $\mathbf{x}_1, \mathbf{x}_2$ instead.

2. A random challenge q is sampled.

3. The sender proves to the receiver that

$$\langle (q, 1), (\underset{\substack{\uparrow \\ (x_1, g)}}{\mathbf{x}_1[1]}, \mathbf{x}_1[2]) \rangle = p_1, \quad \langle (q, 1), (\underset{\substack{\uparrow \\ (x_2, g)}}{\mathbf{x}_2[1]}, \mathbf{x}_2[2]) \rangle = p_2 \,.$$

   The receiver accepts if $p_1 = p_2$.

Similarly, when $\overset{x_1}{\downarrow}\mathbf{x}_1[1] \neq \overset{x_2}{\downarrow}\mathbf{x}_2[1]$, the probability $p_1 = p_2$ is no more than $1/|\mathbb{F}|$, due to the randomness of challenge q. No information about x_1, x_2 are leaked if the sender is proving a true statement, as the leakage p_1 is one-time padded by g, and p_2 is determined by $p_2 = p_1$.

Putting pieces together. We develop the natural idea of using parallel sessions of the protocol for 2MultPlus. Each coordinate of $\hat{f}$ will be computed by a modified version of the 2-round LLW protocol for 2MultPlus (Fig. 4).

In the first round of the LLW protocol for 2MultPlus, party P_1 sends $c_1 := x_1 + a_1$, party P_2 sends $c_2 := x_2 + a_2$, where x_1, x_2 are their inputs, and a_1, a_2 are parts of the scalar OLE correlated randomness. Here c_i is essentially a commitment to x_i.

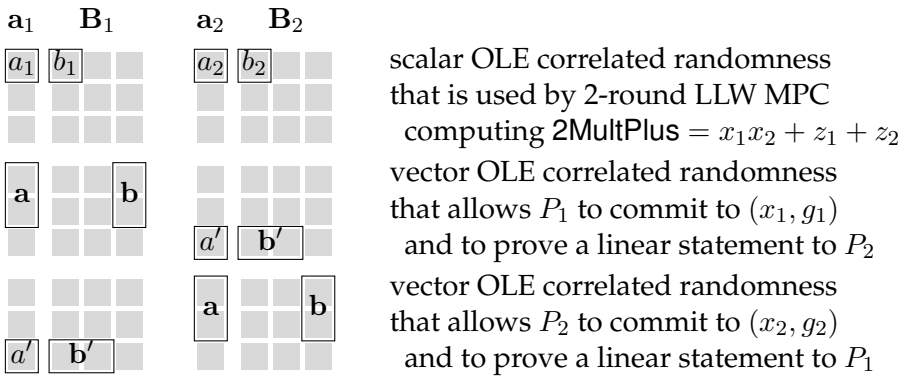

Figure 6: Multiple roles of $\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}_2, \mathbf{B}_2$

In the modified LLW protocol for 2MultPlus, the two parties use 3×3 tensor OLE correlated randomness. That is, party P_1 receives $\mathbf{a}_1 \in \mathbb{F}^3, \mathbf{B}_1 \in \mathbb{F}^{3\times 3}$, party P_2 receives $\mathbf{a}_2 \in \mathbb{F}^3, \mathbf{B}_2 \in \mathbb{F}^{3\times 3}$, where $\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}_2, \mathbf{B}_2$ are random subject to $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$.[6] In the first round, P_1 broadcasts

$$\mathbf{c}_1 := (x_1, g_1) + \mathbf{a}_1[1{:}2],$$

where $\mathbf{a}_1[1{:}2]$ denotes the first two coordinates of $\mathbf{a}_1$. The choice of g_1 will be discussed later. Symmetrically, party P_2 broadcasts $\mathbf{c}_2 := (x_2, g_2) + \mathbf{a}_2[1{:}2]$. The two commitments $\mathbf{c}_1, \mathbf{c}_2$ will have multiple roles:

Compute 2MultPlus. Their first coordinates $\mathbf{c}_1[1], \mathbf{c}_2[1]$ are of the same form as the first message in the original LLW protocol. In the second round, P_1, P_2 will proceed with the original LLW protocol by taking $\mathbf{c}_1[1], \mathbf{c}_2[1]$ as the first round messages.

P_1 proves consistency. Party P_1 may need to prove that the same x_1 is used across different sessions. Note that $\mathbf{c}$ can be viewed as a commitment in the commit-and-prove-linear scheme. P_1 takes $\mathbf{a}_1[1{:}2], \mathbf{B}_1[1{:}2, 3]$ and P_2 takes $\mathbf{a}_2[3], \mathbf{B}_2[3, 1{:}2]$ as the vector OLE correlated randomness (as shown in Fig. 6). Then the ProveSame sub-protocol can prove P_1 is using a consistent value. P_1 sets g_1 according to the sub-protocol.

P_2 proves consistency. Symmetric to the above.

To resolve the well-formedness concern, every party has to prove that it honestly evaluates the preprocessing function. Since the preprocessing function only has multiplication gates, a party can prove well-formedness by using the ProveProd sub-protocol. (Together with ProveSame sub-protocol. Because once a party proves $x_1x_2 = x_3$ using ProveProd, it also need to prove that the "x_1" in ProveProd is the same the "x_1" it uses in 2MultPlus sessions.)

2.4 Lift Security with Output Substitution

Security with output substitution is a weak notion of security. The adversary, after learning the function output y, may adaptively choose y' so that all honest parties (unanimously) output y'. Once we constructed MPC for **P/poly** that is secure with output substitution against malicious corruptions, we can lift its security to the standard notion of security with (unanimous) abort,

[6] Note the transpose of $\mathbf{B}_2$. This makes the equation remains unchanged upon exchanging subscripts.

with the help of *consensus MAC*, which is introduced in [ACGJ19] and will be discussed in Sec. 8. We claim that the following protocol is secure with abort.

1. Party P_i has input x_i, and samples MAC key k_i.
2. Using a protocol that is secure with output substitution to compute
$$(y, \sigma) := f(x_1, \ldots, x_n), \mathsf{Sign}_{k_1,\ldots,k_n}(f(x_1, \ldots, x_n)) .$$
3. Party P_i outputs y if π is a valid signature on y w.r.t. key k_i; aborts otherwise.

As the protocol suggested, consensus MAC has i) a signing algorithm Sign, which takes a message, n keys, generates a signature; and ii) a verification algorithm Verify, which takes a message, a signature and a key, outputs "accept" or "reject".

In the protocol, the adversary, after learning (y, σ), may adaptively replace the output by $(y', \sigma') \neq (y, \sigma)$. In order to achieve security with unanimous abort, all honest parties should reject (y', σ'). This hints how to define consensus MAC: the adversary wins the following game with negligible probability.

1. The adversary chooses the set of corrupted parties $C \subseteq [n]$, and chooses key k_i for each corrupted party $P_i \in C$. Every honest party $P_i \notin C$ samples k_i.
2. The adversary chooses message y and learns $\pi = \mathsf{Sign}_{k_1,\ldots,k_n}(y)$.
3. The adversary adaptively chooses $(y', \pi') \neq (y, \pi)$.
4. The adversary wins if $\mathsf{Verify}_{k_i}(y', \pi') \to$ accept for some honest party $P_i \notin C$.

Here is a simple consensus MAC scheme, whose security will be proven in Sec. 8. Party P_i samples two random number a_i, b_i and let key $k_i := (a_i, b_i)$. The signature $\mathsf{Sign}_{k_1,\ldots,k_n}(y)$ is the degree-n polynomial π such that
$$\pi(0) = y, \quad \pi(a_1) = b_1, \quad \ldots, \quad \pi(a_n) = b_n .$$
The verification accepts a message-signature pair (y, π) w.r.t. key $k_i = (a_i, b_i)$ if and only if $\pi(0) = y$ and $\pi(a_i) = b_i$.

2.5 Tensor OLE Correlated Randomness Generation from OT

We require tensor OLE correlated randomness over a large boolean extension field $\mathbb{F} = \mathsf{GF}(2^\lambda)$, where λ is the security parameter. To generate OLE correlated randomness, it suffices to implement a protocol computing tensor OLE
$$\mathsf{TOLE}((\mathbf{a}, \mathbf{B}), \mathbf{x}) := \mathbf{a}\mathbf{x}^\intercal + \mathbf{B}$$
or random tensor OLE, and later randomize the input-output tuple.

The starting point is a semi-honest protocol [Gil99]. Say the sender has vector $\mathbf{a}$, matrix $\mathbf{B}$; the receiver has vector $\mathbf{x}$ and should learn $\mathbf{Y} = \mathbf{a}\mathbf{x}^\intercal + \mathbf{B}$. Note that $\mathbf{a}\mathbf{x}^\intercal + \mathbf{B}$, as a function in $\mathbf{x}$, is affine over $\mathsf{GF}(2)$. That is, if we let subscript $_{(2)}$ denote bit representations, there exists a binary matrix $M_{\mathbf{a}}$ such that $(\mathbf{Y})_{(2)} = (\mathbf{a}\mathbf{x}^\intercal + \mathbf{B})_{(2)} = M_{\mathbf{a}} \cdot (\mathbf{x})_{(2)} + (\mathbf{B})_{(2)}$. Thus the receiver can learn $\mathbf{Y}$ from OT.

Such protocol is not secure against a malicious sender, who can choose an arbitrary M_1 and let the receiver learn $(\mathbf{Y})_{(2)} = M_1 \cdot (\mathbf{x})_{(2)} + (\mathbf{B})_{(2)}$. To detect malicious behaviour, the receiver samples

a random matrix $\mathbf{Y}^{\text{shadow}}$ and let the sender learn $\mathbf{B}^{\text{shadow}} = \mathbf{a}\mathbf{x}^\intercal + \mathbf{Y}^{\text{shadow}}$. Formally, the sender learns

$$(\mathbf{B}^{\text{shadow}})_{(2)} = M_2 \cdot (\mathbf{a})_{(2)} + (\mathbf{Y}^{\text{shadow}})_{(2)}$$

from OT, where $M_2 = M_{\mathbf{x}}$ if the receiver is honest. The receiver samples a random matrix H over $\mathsf{GF}(2)$, sends H to the sender as a challenge, and asks the sender to guess $H \cdot (\mathbf{Y} + \mathbf{Y}^{\text{shadow}})_{(2)}$. Note that, if both parties are honest, $H \cdot (\mathbf{Y} + \mathbf{Y}^{\text{shadow}})_{(2)} = H \cdot (\mathbf{B} + \mathbf{B}^{\text{shadow}})_{(2)}$. If the sender is corrupted,

$$\begin{aligned} H \cdot (\mathbf{Y} + \mathbf{Y}^{\text{shadow}})_{(2)} &= H \cdot \left(M_1 \cdot (\mathbf{x})_{(2)} + (\mathbf{B})_{(2)} + M_{\mathbf{a}} \cdot (\mathbf{x})_{(2)} + (\mathbf{B}^{\text{shadow}})_{(2)} \right) \\ &= H \cdot (M_1 + M_{\mathbf{a}}) \cdot (\mathbf{x})_{(2)} + H \cdot (\mathbf{B} + \mathbf{B}^{\text{shadow}})_{(2)} \,. \end{aligned}$$

Thus the sender will not be caught if and only if he can guess $H \cdot (M_1 + M_{\mathbf{a}}) \cdot (\mathbf{x})_{(2)}$ correctly. Let H has λ rows and assume w.l.o.g. that the receiver samples $\mathbf{x}$ at random. Then the sender will be caught will overwhelming probability if $\mathrm{rank}(M_1 + M_{\mathbf{a}}) \geq \lambda$. Because in such case, $H \cdot (M_1 + M_{\mathbf{a}}) \cdot (\mathbf{x})_{(2)}$ has large entropy conditioning on the sender's knowledge.

If $\mathrm{rank}(M_1 + M_{\mathbf{a}}) < \lambda$, say we give $(M_1 + M_{\mathbf{a}}) \cdot (\mathbf{x})_{(2)}$ to the corrupted sender, this may only help the adversary. The corrupted sender can compute $(\mathbf{B}')_{(2)} = (M_1 + M_{\mathbf{a}}) \cdot (\mathbf{x})_{(2)} + (\mathbf{B})_{(2)}$. (Honest sender simply let $\mathbf{B}' = \mathbf{B}$.) Note that

$$\mathbf{Y} = \mathbf{a}\mathbf{x}^\intercal + \mathbf{B}' \,.$$

The sender and receiver output $(\mathbf{a}, \mathbf{B}'), (\mathbf{x}, \mathbf{Y})$ respectively.

Such a protocol is insecure, for two reasons.

- If the sender is corrupted, he additionally knows $(M_1 + M_{\mathbf{a}}) \cdot (\mathbf{x})_{(2)}$, which is an at most λ-bit leakage of $\mathbf{x}$. If the receiver is corrupted, she additionally knows

  $$H \cdot (\mathbf{B} + \mathbf{B}^{\text{shadow}})_{(2)} := H \cdot (M_2 + M_{\mathbf{b}}) \cdot (\mathbf{a})_{(2)},$$

  which is an at most λ-bit leakage of $\mathbf{a}$. The leakage can be removed by using randomness extractor and left-over hash lemma.

- The corrupted sender (resp. receiver) can arbitrarily choose $\mathbf{a}, \mathbf{B}$ (resp. $\mathbf{x}$). To ensure randomness, an additional re-randomization step is needed.

3 Definitions

For any positive integer n, let $[n] := \{1, 2, \ldots, n\}$. Let $\mathsf{GF}(p)$ be the finite field of order p. A finite field is typically denoted by $\mathbb{F}$. For scalars $v_1, v_2, \ldots, v_n \in \mathbb{F}$, the vector $(v_1, \ldots, v_n) \in \mathbb{F}^n$ may also be denoted by $v_1 \| v_2 \| \ldots \| v_n$. A vector is typically denoted by a boldface lowercase letter. For a vector $\mathbf{a}$, let $\mathbf{a}[i]$ denote its i-th entry, let $\mathbf{a}[i{:}i']$ denote the sub-vector $(\mathbf{a}[i], \mathbf{a}[i+1], \ldots, \mathbf{a}[i'])$. For two vectors $\mathbf{a}, \mathbf{b}$ of the same dimension, let $\langle \mathbf{a}, \mathbf{b} \rangle := \mathbf{a}^\intercal \mathbf{b}$ denote the inner product of $\mathbf{a}, \mathbf{b}$. A matrix is typically denoted by a boldface capital letter. For a matrix $\mathbf{M}$, let $\mathbf{M}[i, j]$ denote its entry across the i-th row and the j-column; let $\mathbf{M}[i{:}i', j]$ denote the vector $(\mathbf{M}[i, j], \mathbf{M}[i+1, j], \ldots, \mathbf{M}[i', j])$, which a sub-vector of the j-th column of $\mathbf{M}$; let $\mathbf{M}[i, j{:}j']$ denote the vector $(\mathbf{M}[i, j], \mathbf{M}[i, j+1], \ldots, \mathbf{M}[i, j'])$, which a sub-vector of the i-th row of $\mathbf{M}$.

Tensor OLE is a natural extension of scalar OLE and vector OLE. The sender has vector $\mathbf{a}$, matrix $\mathbf{B}$; the receiver has vector $\mathbf{b}$. After the interaction, the receiver learns $\mathbf{a}\mathbf{b}^\intercal + \mathbf{B}$. Similarly, we can define tensor OLE correlated randomness.

Definition 1. The $\ell_1 \times \ell_2$ tensor OLE correlated randomness over field $\mathbb{F}$ is the distribution of $((\mathbf{a}_1, \mathbf{B}_1), (\mathbf{a}_2, \mathbf{B}_2))$, where $\mathbf{a}_1 \in \mathbb{F}^{\ell_1}, \mathbf{B}_1 \in \mathbb{F}^{\ell_1 \times \ell_2}, \mathbf{a}_2 \in \mathbb{F}^{\ell_2}, \mathbf{B}_2 \in \mathbb{F}^{\ell_2 \times \ell_1}$ are uniformly random subject to $\mathbf{a}_1 \mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2$.

The tensor OLE correlated randomness degenerates into vector OLE correlated randomness if $\ell_2 = 1$, and into scalar OLE correlated randomness if $\ell_1 = \ell_2 = 1$.

3.1 Secure Multi-Party Computation

Definition 2 (MPC Protocol with Broadcast and P2P Channels)**.** An r-rounds MPC protocol Π for a n-party function $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{Y}$ consists of n next-step algorithms $(\mathsf{Next}_i)_{i \in [n]}$. An execution of Π with inputs $(x_1, \ldots, x_n) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ and security parameter 1^λ proceeds as follows:

Randomness: Each party P_i samples local randomness $r_i \leftarrow \mathcal{R}_i$ from its local randomness space $\mathcal{R}_i$. It initializes its state as $\mathsf{st}_i^{(0)} = (x_i, r_i)$.

Round $1 \le j \le r$**:** Every party P_i computes the next-round messages from its current state $(m_i^{(j)}, m_{i \to 1}^{(j)}, \cdots, m_{i \to n}^{(j)}) \leftarrow \mathsf{Next}_i(1^\lambda, \mathsf{st}_i^{(j-1)})$, broadcasts $m_i^{(j)}$, and sends message $m_{i \to i'}^{(j)}$ to party $P_{i'}$ for every $i' \in [n]$. Party P_i receives message $m_{i'}^{(j)}, m_{i' \to i}^{(j)}$ from party $P_{i'}$, updates its state as $\mathsf{st}_i^{(j)} = (\mathsf{st}_i^{(j-1)}, (m_{i'}^{(j)}, m_{i' \to i}^{(j)})_{i' \in [n]})$.

Output: After r rounds, every party P_i computes $y_i \leftarrow \mathsf{Next}_i(1^\lambda, \mathsf{st}_i^{(r)})$, and outputs y_i.

We also consider MPC protocol that relies on **correlated randomness**. There is a joint distribution $\mathcal{D}$ over $\mathcal{R}'_1 \times \cdots \times \mathcal{R}'_n$. In the execution of the protocol, correlated randomness $(r'_1, \ldots, r'_n) \leftarrow \mathcal{D}$ is sampled by the beginning of the protocol, and party P_i's state is initialized as $\mathsf{st}_i^{(0)} = (x_i, r_i, r'_i)$.

The efficiency of the protocol is mainly measured by its **communication complexity**, including the total length of messages sent in P2P channels and the total length of messages broadcast. We also consider the **correlated randomness complexity**. Since we are going to use independent samples of a 2-ary correlated randomness called OLE correlated randomness, the correlated randomness complexity can be measured how many instances of OLE correlated randomness are sampled.

Below, we suppress the appearance of the security parameter 1^λ, which is assumed implicitly.

Common Output. We remark that the above definition assumes all parties get the same output from the protocol. It can be generalized to the case where each party learns a different output. From a protocol design point of view, it is without loss of generality to consider a common output: To compute function f mapping $x_1, \ldots, x_n$ to different outputs $y_1, \ldots, y_n$, every party P_i can sample a one-time pad k_i of appropriate length and jointly compute the augmented function mapping $(x_1, k_1), \ldots, (x_n, k_n)$ to $(y_1 + k_1, \ldots, y_n + k_n)$, where k_i's and $+$ should be defined appropriately for the specific function f. For instance, if f is a Boolean computation, k_i's should be random strings and $+$ is XOR, and if f is an arithmetic computation over a finite field, k_i's should be random vectors and $+$ over the field.

Public Output Reconstruction. The output of a MPC protocol is publicly reconstructible if the output of every party is only determined by broadcast messages. Public output reconstruction naturally associate with common output, as it allows any party (or even a bystander) to learn the outputs of all parties.

Model for Protocol Execution. The real execution $\mathsf{Exec}_{C,\mathcal{A},\mathcal{Z}}(1^\lambda, \Pi)$ is defined by running the protocol Π with a non-uniform interactive Turing machine $\mathcal{A}$ (called the adversary), and an interactive Turing machine $\mathcal{Z}$ (called the environment). We consider static corruption. The adversary $\mathcal{A}$ corrupts a subset of the n parties, denoted by $C \subseteq [n]$ at the beginning. The adversary $\mathcal{A}$ controls the corrupted parties $\{P_i\}_{i\in C}$ and can communicate arbitrarily with the environment $\mathcal{Z}$. We consider malicious adversary $\mathcal{A}$ who can deviate from the protocol specification, while the honest parties $\{P_i\}_{i\notin C}$ follow the protocol. The inputs of the honest parties are chosen by $\mathcal{Z}$ adaptively, and as soon as an honest party produces an output, $\mathcal{Z}$ learns the output. The adversary is rushing: in each round, it waits for all the messages from the honest parties before sending any message. The output of the experiment is the output of the environment $\mathcal{Z}$. We overload the notation to also use $\mathsf{Exec}_{C,\mathcal{A},\mathcal{Z}}(1^\lambda, \Pi)$ to denote the distribution of the output of the experiment.

Ideal Functionality $\mathcal{F}$. The ideal functionality should be viewed as a trusted external party. It is modeled as an interactive Turing machine that interacts with the parties and the adversary. For example, the ideal functionality $\mathcal{F}_f$ receives an input x_i from each party P_i, then sends $f(x_1, \ldots, x_n)$ to all parties. We will define ideal functionality $\mathcal{F}$ who also interacts with the adversary according to its code.

Assuming the ideal functionality $\mathcal{F}$ exists, there is a canonical protocol where everyone only talks to $\mathcal{F}$: All parties simply hand their inputs to $\mathcal{F}$. The adversary may interacts with $\mathcal{F}$ if $\mathcal{F}$ has any "backdoor interface". Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. Such protocol, denoted by $\Pi_\mathcal{F}$, is call the dummy protocol for functionality $\mathcal{F}$. And the parties in the dummy protocol are called dummy parties.

Ideal Execution and Simulation. The ideal execution of the protocol $\Pi_\mathcal{F}$ with a simulator $\mathcal{S}$, and environment $\mathcal{Z}$ is $\mathsf{Exec}_{C,\mathcal{S},\mathcal{Z}}(1^\lambda, \Pi_\mathcal{F})$.

In this work, we will use the typical black-box and straight-line simulation, in which $\mathcal{S}$ interacts with the real-world adversary $\mathcal{A}$, forwarding its communication with $\mathcal{Z}$, and simulating messages of honest parties of Π for $\mathcal{A}$. Below, we detail on the ideal execution with such a simulator, and denote it as $\mathsf{Ideal}_{C,\mathcal{A},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathcal{F})$.

$\mathcal{Z}$ chooses inputs of honest dummy parties: For each $i \notin C$, the input of honest dummy party P_i running protocol $\Pi_\mathcal{F}$ is chosen adaptively by $\mathcal{Z}$, and Π immediately sends the the input to $\mathcal{Z}$. And as soon as $\mathcal{F}$ outputs a value to an honest dummy party, the output is learnt by $\mathcal{Z}$.

$\mathcal{S}$ simulates the real execution $\mathsf{Exec}_{C,\mathcal{A}}(1^\lambda, \Pi)$ for $\mathcal{A}$: The simulator $\mathcal{S}$ forwards all communication between $\mathcal{A}$ and $\mathcal{Z}$, and simulates an execution of the protocol Π for $\mathcal{A}$ in the following way. All messages that $\mathcal{A}$ sends corresponding to corrupted parties (in C) running Π are received by $\mathcal{S}$, and $\mathcal{S}$ simulates messages from the honest parties running Π for $\mathcal{A}$.

$\mathcal{S}$ interacts with $\mathcal{F}$: The simulator $\mathcal{S}$ interacts with ideal functionality $\mathcal{F}$ as the corrupted parties. In particular, it chooses the input of every dummy corrupted party P_i for $i \in C$, and as soon as $\mathcal{F}$ outputs a value to P_i, the output is learnt by $\mathcal{S}$. In addition, depending on the specification of $\mathcal{F}$, it may directly communicate with $\mathcal{S}$.

Output: The output of the experiment is the output of the environment $\mathcal{Z}$.

We overload the notation to also use $\mathsf{Ideal}_{C,\mathcal{A},\mathcal{S},\mathcal{Z}}(1^\lambda, \Pi)$ to denote the distribution of the output of the experiment.

Definition 3. Let $\mathcal{F}$ be a n-party functionality. Let Π be a MPC protocol. Protocol Π computationally (resp. statistically) securely implements $\mathcal{F}$ against malicious corruptions if for any non-

uniform poly-time (resp. unbounded) interactive Turing machine $\mathcal{A}$, there exists an uniform poly-time interactive Turing machine $\mathcal{S}$, such that, for every non-uniform poly-time (resp. unbounded) interactive machine $\mathcal{Z}$,

$$\left|\Pr\left[\mathsf{Exec}_{C,\mathcal{A},\mathcal{Z}}(1^\lambda,\Pi)\to 1\right]-\Pr\left[\mathsf{Ideal}_{C,\mathcal{A},\mathcal{S},\mathcal{Z}}(1^\lambda,\mathcal{F})\to 1\right]\right|\le \mathrm{negl}(\lambda)\,.$$

Figure 7: The ideal functionalities $\mathcal{F}_f^{\mathsf{abort}}$ and $\mathcal{F}_f^{\mathsf{os}}$ capture the computation of function f with different security guarantees

Input: On input $(\texttt{"Input"}, x_i)$ from P_i. The functionality stores (P_i, x_i), and sends $(P_i, \texttt{"Received"})$ to *all parties*.

Output: On input $(\texttt{"Output"})$ from the adversary, the functionality computes $\mathbf{y} = f(x_1, \ldots, x_n)$, and sends $(\texttt{"Output"}, y)$ to *the adversary*.

Abort (only in $\mathcal{F}_f^{\mathsf{abort}}$): On input $(\texttt{"Output"}, 1_{\mathsf{abort}})$ from the adversary, where $1_{\mathsf{abort}} \in \{0,1\}$ is an indicator bit, the functionality sends $(\texttt{"Output"}, \perp)$ to all parties if $1_{\mathsf{abort}} = 1$, sends $(\texttt{"Output"}, y)$ to all parties if $1_{\mathsf{abort}} = 0$.

Output Substitution (only in $\mathcal{F}_f^{\mathsf{os}}$): On input $(\texttt{"Output"}, y')$ from the adversary, the functionality sends $(\texttt{"Output"}, y')$ to all parties.

The ideal functionality $\mathcal{F}_f^{\mathsf{abort}}$ captures the standard notion of "security with unanimous abort". $\mathcal{F}_f^{\mathsf{os}}$ captures a weaker security, in which the adversary learns no extra information compare to an honest execution, but the adversary can choose an incorrect (unanimous) output for all honest parties. A similar security notion "privacy with knowledge of outputs (PKO)" is considered by [IKP10], which allows the adversary to choose incorrect outputs for selected honest parties.

Sec. 8 shows a non-interactive reduction from $\mathcal{F}^{\mathsf{os}}$ to $\mathcal{F}^{\mathsf{abort}}$. That is, given a protocol implementing $\mathcal{F}_{f'}^{\mathsf{os}}$, where f' is induced from f and has similar complexity, there exists a protocol implementing $\mathcal{F}_f^{\mathsf{abort}}$ with the same communication and correlated randomness complexity. Therefore, for a complexity class (e.g. **P/poly**) under which the reduction $f \mapsto f'$ is close, it suffices to construct MPC protocols that is secure with output substitution.

3.2 Multi-Party Randomized Encoding

Let λ be the security parameter. All objects below implicitly depend on λ.

We begin by recalling the definition of randomized encoding. A function f is encoded by a randomized function g, if the output of f (and nothing else) can be recovered from the output of g.

Definition 4 (Randomized Encoding [AIK04])**.** Let $f : \mathcal{X} \to \mathcal{Y}$ be some function. The *randomized encoding* of f is a function $g : \mathcal{X} \times \mathcal{R} \to \hat{\mathcal{Y}}$, where $\mathcal{R}$ is the randomness space, $\hat{\mathcal{Y}}$ is the encoding space. A randomized encoding should be both correct and private.

Correctness There is a decoding function Dec such that for all $x \in \mathcal{X}, r \in \mathcal{R}$, it holds that

$$\mathsf{Dec}(g(x;r)) = f(x).$$

Privacy There exists an efficient randomized simulation algorithm $\mathcal{S}$ such that for any $x \in \mathcal{X}$, the distribution of $\mathcal{S}(f(x))$ is identical to that of $g(x;r)$. The privacy can be relaxed to statistical privacy (resp. computational privacy), if the $\mathcal{S}(f(x))$ and $g(x;r)$ are statistically close (resp. computational indistinguishable).

The task of computing a function f can be non-interactively reduced to the task of computing its randomized encoding g. Given a protocol Π_g for computing

$$(x_1, r_1), \ldots, (x_n, r_n) \mapsto g(x_1, \ldots, x_n; r_1 + \cdots + r_n),$$

it directly induces a protocol Π_f for computing f: party P_i locally samples randomness r_i; jointly compute the encoding by executing protocol Π_g; then every party locally decodes the output. It is easy to show that Π_f satisfies the same security notion as Π_g.

The technique is generalized to *multi-party randomized encoding (MPRE)* in pursuit of round-optimal semi-honest MPC protocols [ABT18]. The maliciously secure analog of MPRE is consider in [ABT19]. We recall the definition of multi-party randomized encoding (MPRE). In addition, we extend the definition to incorporate with correlated randomness and semi-malicious security.

Definition 5 (Multi-Party Randomized Encoding)**.** Let $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{Y}$ be some n-party function. A *multi-party randomized encoding* (MPRE) of f is specified by

- Local randomness space $\mathcal{R}_i$ for $i \in [n]$. Correlated randomness space $\mathcal{R}'_1 \times \cdots \times \mathcal{R}'_n$ together with a distribution $\mathcal{D}$ over it.
- Local preprocessing function $h_i : \mathcal{X}_i \times \mathcal{R}_i \times \mathcal{R}'_i \to \hat{\mathcal{X}}_i$.
- Encoding function $\hat{f} : \hat{\mathcal{X}}_1 \times \cdots \times \hat{\mathcal{X}}_n \to \hat{\mathcal{Y}}$.
- Decoding function $\mathsf{Dec} : \hat{\mathcal{Y}} \to \mathcal{Y}$.

Such that for any input $(x_1, \ldots, x_n)$, the encoding

$$\hat{y} := \hat{f}\big(h_1(x_1, r_1, r'_1), \ldots, h_n(x_n, r_n, r'_n)\big) \tag{3}$$

represents $y = f(x_1, \ldots, x_n)$ in the following sense:

Correctness For any input $(x_1, \ldots, x_n) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, randomness $(r_1, \ldots, r_n) \in \mathcal{R}_1 \times \cdots \times \mathcal{R}_n$ and correlated randomness $(r'_1, \ldots, r'_n)$ in the support of $\mathcal{D}$, the corresponding encoding $\hat{y}$ defined by (3) satisfies that $f(x_1, \ldots, x_n) = \mathsf{Dec}(\hat{y})$.

Semi-Malicious Security We say MPRE is computationally (resp. statistically) semi-malicious secure with output substitution, if the following canonical protocol computationally (resp. statistically) securely implements $\mathcal{F}_f^{\mathsf{os}}$:

- The protocol assumes ideal functionality $\mathcal{F}_{\hat{f} \circ h}^{\mathsf{os}}$.
- On input x_i, party P_i samples $r_i \leftarrow \mathcal{R}_i$ locally, receives r'_i where $(r'_1, \ldots, r'_n)$ is sampled from $\mathcal{D}$, then inputs (x_i, r_i, r'_i) to the ideal functionality $\mathcal{F}_{\hat{f} \circ h}^{\mathsf{os}}$. Note that a corrupted party may adaptively modify its input of $\mathcal{F}_{\hat{f} \circ h}^{\mathsf{os}}$.
- Upon receiving $\hat{y}$ from $\mathcal{F}_{\hat{f} \circ h}^{\mathsf{os}}$, party decodes $y = \mathsf{Dec}(\hat{y})$, and outputs y.

Malicious Security We say MPRE is computationally (resp. statistically) malicious secure with output substitution, if the following canonical protocol computationally (resp. statistically) securely implements $\mathcal{F}_f^{\mathsf{os}}$:

- The protocol assumes ideal functionality $\mathcal{F}_{\hat{f}}^{\mathsf{os}}$.
- On input x_i, party P_i samples $r_i \leftarrow \mathcal{R}_i$ locally, receives r'_i where $(r'_1, \ldots, r'_n)$ is sampled from $\mathcal{D}$, and locally computes $\hat{x}_i = h_i(x_i, r_i, r'_i)$ then inputs $\hat{x}_i$ to the ideal functionality $\mathcal{F}_{\hat{f}}^{\mathsf{os}}$.

- Upon receiving $\hat{y}$ from $\mathcal{F}_{\hat{f}}^{\mathsf{os}}$, party decodes $y = \mathsf{Dec}(\hat{y})$, and outputs y.

We expand the definition of semi-malicious MPRE in more detail by describing the ideal world and real world of the security game of the canonical protocol (Fig. 8). Apparently, the adversary and the environment learn no information during the last two steps in the security game.

The Real Execution $\mathsf{Exec}_{C,\mathcal{A},\mathcal{Z}}(1^\lambda, \Pi)$

1. The adversary $\mathcal{A}$ receives the corrupted parties' portions of correlated randomness $\{r'_i\}_{i\in C}$.
2. The environment $\mathcal{Z}$ chooses the input x_i for each honest party $P_i \notin C$. Every honest P_i samples local randomness r_i, receives his portion of correlated randomness r'_i and sends (x_i, r_i, r'_i) to $\mathcal{F}_{\hat{f}\circ h}^{\mathsf{os}}$.
3. A corrupted party $P_i \in C$ may input any $(\bar{x}_i, \bar{r}_i, \bar{r}'_i)$ to $\mathcal{F}_{\hat{f}\circ h}^{\mathsf{os}}$.
4. $\mathcal{F}_{\hat{f}\circ h}^{\mathsf{os}}$ sends the output $\hat{y}$ to $\mathcal{A}$. The adversary $\mathcal{A}$ chooses and sends $\hat{y}'$ back.
5. Every honest party receives $\hat{y}'$ from $\mathcal{F}_{\hat{f}\circ h}^{\mathsf{os}}$, and output $y' = \mathsf{Dec}(\hat{y}')$ to $\mathcal{Z}$.

The Ideal Execution $\mathsf{Ideal}_{C,\mathcal{A},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathcal{F}_f^{\mathsf{os}})$

1. The simulator $\mathcal{S}$ receives the corrupted parties' portions of correlated randomness $\{r'_i\}_{i\in C}$, and forwards them to $\mathcal{A}$.
2. The environment $\mathcal{Z}$ chooses the input x_i for each honest party $P_i \notin C$. Every honest dummy P_i directly inputs x_i to $\mathcal{F}_f^{\mathsf{os}}$.
3. Any corrupted party's input to $\mathcal{F}_{\hat{f}\circ h}^{\mathsf{os}}$ is hijacked by $\mathcal{S}$. The simulator $\mathcal{S}$ extracts input x_i for each $i \in C$, and sends x_i to $\mathcal{F}_f^{\mathsf{os}}$ on behalf of dummy P_i.
4. $\mathcal{F}_f^{\mathsf{os}}$ sends the output y to $\mathcal{S}$, who generates and sends $\hat{y}$ to $\mathcal{A}$. The adversary $\mathcal{A}$ chooses and sends $\hat{y}'$ back to $\mathcal{S}$, who sends $y' = \mathsf{Dec}(\hat{y}')$ to $\mathcal{F}_f^{\mathsf{os}}$.
5. Every honest dummy party receives y' from $\mathcal{F}_f^{\mathsf{os}}$, and output y' to $\mathcal{Z}$.

Figure 8: The Security Game of Semi-Malicious MPRE.

Effective Degree. By definition, the task of computing f against malicious corruptions is reduced to the task of computing $\hat{f}$, if MPRE is maliciously secure. To minimize the round complexity for computing $\hat{f}$, the classical approach is to reduce the arithmetic degree of $\hat{f}$. The degree of $\hat{f}$ is called the *effective degree* of this MPRE.

Formally, let $\mathbb{F}$ be a finite field. Let

$$\left(\hat{f} : \hat{\mathcal{X}}_1 \times \cdots \times \hat{\mathcal{X}}_n \to \hat{\mathcal{Y}}, h_1, \ldots, h_n\right)$$

be a MPRE for f, such that $\hat{\mathcal{X}}_1, \ldots, \hat{\mathcal{X}}_n, \hat{\mathcal{Y}}$ are vector spaces over field $\mathbb{F}$. The arithmetic degree of $\hat{f}$ over $\mathbb{F}$ is called the effective degree of the MPRE.

Arithmetic Preprocessing. By definition, the task of computing f against malicious corruptions is reduced to computing $\hat{f}\circ h$ if MPRE is semi-maliciously secure. To minimize the round complexity for computing $\hat{f} \circ h$, besides reducing the degree of $\hat{f}$, we also need the preprocessing functions $h_1, \ldots, h_n$ to be computable by poly-size arithmetic circuits.

Formally, let $(\hat{f}, h_1, \ldots, h_n)$ be a MPRE for f, who has low effective degree over a field $\mathbb{F}$. The MPRE has arithmetic preprocessing if its input spaces, local randomness spaces and correlated randomness spaces are vector spaces over $\mathbb{F}$, and the local preprocessing functions $h_1, \ldots, h_n$ are computed by poly-size arithmetic circuits over $\mathbb{F}$.

If a MPRE has effective degree 2 and arithmetic preprocessing over $\mathbb{F}$, we say $\hat{f}\circ g$ is an *effective-degree-2 function* over $\mathbb{F}$.

Definition 6 (Effective-Degree-2 Function)**.** A function $g : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{Y}$ is of *effective degree 2* over a field $\mathbb{F}$, if $\mathcal{X}_1, \ldots, \mathcal{X}_n, \mathcal{Y}$ are vector spaces over $\mathbb{F}$ and there exist

- $h_i : \mathcal{X}_i \to \hat{\mathcal{X}}_i$, an arithmetic circuit over $\mathbb{F}$, for each $i \le n$.
- $\hat{f} : \hat{\mathcal{X}}_1 \times \cdots \times \hat{\mathcal{X}}_n \to \mathcal{Y}$, a degree-2 arithmetic function over $\mathbb{F}$.

such that for all $(x_1, \ldots, x_n) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$,

$$\hat{f}(h_1(x_1), \ldots, h_n(x_n)) = g(x_1, \ldots, x_n).$$

4 MPRE for Degree-3 Functions

Let λ be the security parameter. All objects implicitly depend on λ. Most objects in this section is arithmetic over a finite field $\mathbb{F} = \mathbb{F}(\lambda)$, such that $|\mathbb{F}| = \Omega(2^\lambda)$.

Canonical Form Polynomials. Before we construct MPRE for degree-3 functions, we observe that it is w.l.o.g. to assume the degree-3 function f is of the following canonical form.

For any **canonical degree-3 function** $f : \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_n} \to \mathbb{F}^{\ell}$, there is an index set $\mathcal{I} \subseteq [\ell]$, such that for each $t \in [\ell]$,

- If $t \notin \mathcal{I}$, the t-th coordinate of f, denoted by f_t, is of degree at most 2.
- If $t \in \mathcal{I}$, then $f_t = x_1x_2x_3 + z_1 + z_2 + z_3$ where x_i, z_i are from the same party.

  More formally, let $\mathbf{x}_i \in \mathbb{F}^{\ell_i}$ denote the i-th input of f, then

  $$f_t(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \mathbf{x}_{i_{t,1}}[j_{t,1}] \cdot \mathbf{x}_{i_{t,2}}[j_{t,2}] \cdot \mathbf{x}_{i_{t,3}}[j_{t,3}] + \mathbf{x}_{i_{t,1}}[j'_{t,1}] + \mathbf{x}_{i_{t,2}}[j'_{t,2}] + \mathbf{x}_{i_{t,3}}[j'_{t,3}]$$

  for some $i_{t,1}, i_{t,2}, i_{t,3} \in [n]$ and $j_{t,1}, j'_{t,1} \in [\ell_{i_{t,1}}], \ldots, j_{t,3}, j'_{t,3} \in [\ell_{i_{t,3}}]$.

We assume w.l.o.g. that f is a canonical degree-3 function. It is known in the literature that (e.g. shown by [BGI+18, GIS18, LLW20]) every degree-3 function f has a semi-honest MPRE whose encoding function is canonical. The MPRE does not use correlated randomness, and is perfectly secure, thus it is also semi-maliciously secure. The MPRE does not has preprocessing (preprocessing functions are identity functions), thus semi-malicious security implies malicious security.

Moreover, such canonicalization does not increase complexity. Remind that the complexity measure we care about is its total number of monomials $\mathsf{mc}(f)$. It is not difficult to show that $\mathsf{mc}(\hat{f}) = O(\mathsf{mc}(f))$, where $\hat{f}$ is the encoding function of the MPRE for f we just discussed.

We will use the same canonicalization for degree-2 functions in Sec. 5.

4.1 Background: Semi-honest MPRE for Degree-3 Functions

Due to canonicalization, it is sufficient to consider the minimal complete function

$$\mathsf{3MultPlus}\Big((x_1, z_1), (x_2, z_2), (x_3, z_3)\Big) = x_1x_2x_3 + z_1 + z_2 + z_3.$$

It only involves three parties P_1, P_2, P_3. Party P_i has input $x_i, z_i \in \mathbb{F}$. The output can also be presented as a $\mathbb{F}$-modular branching program:

$$x_1x_2x_3 + z_1 + z_2 + z_3 = \det \begin{bmatrix} x_1 & & z_1 + z_2 + z_3 \\ -1 & x_3 & \\ & -1 & x_2 \end{bmatrix}$$

As shown by AIK, it has a degree-3 random encoding

$$\begin{bmatrix} 1 & a_1 & a_4 \\ & 1 & \\ & & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 & & z_1+z_2+z_3 \\ -1 & x_3 & \\ & -1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & a_3 & a_5 \\ & 1 & a_2 \\ & & 1 \end{bmatrix}$$
$$= \begin{bmatrix} x_1 - a_1 & \begin{pmatrix} a_3x_1 + a_1x_3 \\ -a_1a_3 - a_4 \end{pmatrix} & \begin{pmatrix} \boxed{a_1a_2x_3} + a_5x_1 - a_1a_5 + \\ a_4x_2 - a_2a_4 + z_1 + z_2 + z_3 \end{pmatrix} \\ -1 & x_3 - a_3 & a_2x_3 - a_5 \\ & -1 & x_2 - a_2 \end{bmatrix},$$

where $a_1, \ldots, a_5 \in \mathbb{F}$ are the randomness of the encoding.

Notice that the randomized encoding only has one degree-3 monomial term $a_1a_2x_3$ (highlighted by a box). Assume a_1, a_2 are sampled from scalar OLE correlated randomness, that is, $a_1, b_1, a_2, b_2 \in \mathbb{F}$ are randomly sampled conditioning on $a_1a_2 = b_1 + b_2$, then $a_1a_2x_3 = (b_1 + b_2)x_3$ becomes degree-2. This observation is formalized by [LLW20], there is an effective-degree-2 semi-honest MPRE for 3MultPlus using OLE correlated randomness, as presented in Fig. 9.

Figure 9: **The Effective-Degree-2 Semi-Honest MPRE for** 3MultPlus

Input: P_i has $x_i, z_i \in \mathbb{F}$.

Local Randomness: P_1 samples $a_{4,1} \in \mathbb{F}$; P_2 samples $a_{5,2} \in \mathbb{F}$; P_3 samples $a_3, a_{4,3}, a_{5,3} \in \mathbb{F}$.

Correlated Randomness: $a_1, b_1, a_2, b_2 \in \mathbb{F}$ are randomly sampled under constraint $a_1a_2 = b_1 + b_2$. P_1 receives (a_1, b_1). P_2 receives (a_2, b_2). P_3 has no correlated randomness.

Preprocessing: None. I.e., the preprocessing functions are identity functions.

Encoding Function: Outputs the following matrix

$$\widehat{\mathsf{3MultPlus}}\left(\left((x_1,z_1),a_{4,1},(a_1,b_1)\right),\left((x_2,z_2),a_{5,2},(a_2,b_2)\right),\left((x_3,z_3),(a_3,a_{4,3},a_{5,3}),\perp\right)\right) =$$
$$\begin{bmatrix} x_1 - a_1 & \begin{pmatrix} a_3x_1 + a_1x_3 \\ -a_1a_3 - a_4 \end{pmatrix} & \begin{pmatrix} b_1x_3 + b_2x_3 + a_5x_1 - a_1a_5 + \\ a_4x_2 - a_2a_4 + z_1 + z_2 + z_3 \end{pmatrix} \\ -1 & x_3 - a_3 & a_2x_3 - a_5 \\ & -1 & x_2 - a_2 \end{bmatrix}, \quad (4)$$

where $a_4 := a_{4,1} + a_{4,3}$ and $a_5 := a_{5,2} + a_{5,3}$.

Decoding Function: Outputs the determinant of the encoding.

4.2 CDS Encoding

We observe that the MPRE presented in Fig. 9 is not semi-maliciously secure. Semi-malicious security does not follow automatically from perfect semi-honest security because of the correlated randomness. Party P_1 or P_2 can locally modify its portion of the correlated randomness. For example, if corrupted P_1 replaces b_1 by $b_1 + 1$, the decoding will output $x_1x_2x_3 + x_3 + z_1 + z_2 + z_3$. Similarly, if P_1 replaces a_1 by $a_1 + 1$, the decoding will output $x_1x_2x_3 - a_2x_3 + z_1 + z_2 + z_3$. In either case, privacy is lost.

To prevent such attacks, we will invent a tool called *conditional disclosure of secret (CDS) encoding*, which reveals a secret only if a_1, a_2, b_1, b_2 satisfy the OLE relation $a_1a_2 = b_1 + b_2$.

We start with protecting P_3's privacy, the cases of protecting P_1 or P_2 are similar. Party P_1 has $a_1, b_1 \in \mathbb{F}$. Party P_2 has $a_2, b_2 \in \mathbb{F}$. Party P_3 has a secret $s \in \mathbb{F}$. We would like to construct a CDS encoding that achieves three goals:

Figure 10: CDS Encoding

The outer MPRE specifies 3 parties, denoted by P_1, P_2, P_3

Input: Party P_3 receives as input $s \in \mathbb{F}$. Party P_1 receives random $\mathbf{a}_1 \in \mathbb{F}^2, \mathbf{B}_1 \in \mathbb{F}^{2\times 2}$. Party P_2 receives random $\mathbf{a}_2 \in \mathbb{F}^2, \mathbf{B}_2 \in \mathbb{F}^{2\times 2}$. $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{B}_1, \mathbf{B}_2)$ is supposed to be sampled from tensor-OLE correlated randomness.

Randomness: In addition, P_3 samples $q_1, q_2, r_1, r_2 \in \mathbb{F}$.

Preprocessing: P_3 locally computes $r_1q_1, r_2q_2, q_1q_2, r_2q_1, r_2q_1q_2$.

Encoding Function: Define $\mathbf{q}_1 = (q_1, 1), \mathbf{q}_2 = (q_2, 1)$. The functionality outputs $p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle$, $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$, $p_3 = \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1\mathbf{q}_2^\intercal \rangle$ and

$$\mathbf{c} = \begin{bmatrix} 1 & \langle \mathbf{a}_2, \mathbf{q}_2 \rangle \\ \langle \mathbf{a}_1, \mathbf{q}_1 \rangle & \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1\mathbf{q}_2^\intercal \rangle \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} + \begin{bmatrix} 0 \\ s \end{bmatrix}$$

Decoding Function: Check if $p_1p_2 = p_3$. If so, output $\langle \mathbf{c}, (-p_1, 1) \rangle$. Otherwise, output $\perp$.

- To disclose s if $a_1a_2 = b_1 + b_2$.
- To hide s if $a_1a_2 \neq b_1 + b_2$.
- To reveal nothing about a_1, a_2, b_1, b_2 except whether $a_1a_2 = b_1 + b_2$.
- The encoding function is quadratic.

We stress that the CDS encoding we are going to build *is not* a MPRE for any function, since it does not follow the same syntax. For example, the security of CDS encoding will rely on the fact that (a_1, a_2, b_1, b_2) is sampled from OLE correlated randomness. The CDS encoding will be used as a sub-module of the semi-maliciously secure MPRE in Sec. 4.3, where CDS encoding is carefully aligned with the rest of the MPRE.

It turns out that, towards building CDS encoding, we need to replace scalar OLE correlated randomness with tensor OLE correlated randomness. Let P_1 receive random $\mathbf{a}_1 \in \mathbb{F}^2$, $\mathbf{B}_1 \in \mathbb{F}^{2\times 2}$ and let P_2 receive random $\mathbf{a}_2 \in \mathbb{F}^2, \mathbf{B}_2 \in \mathbb{F}^{2\times 2}$, such that $\mathbf{a}_1\mathbf{a}_1^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$. Party P_1, P_2 use the first coordinates of the tensor-OLE correlated randomness as the original scalar OLE correlated randomness. That is, let $(a_1, b_1, a_2, b_2) := (\mathbf{a}_1[1], \mathbf{B}_1[1], \mathbf{a}_2[1], \mathbf{B}_2[1])$. The remaining coordinates will be used to hide a_1, b_1, a_2, b_2.

The CDS encoding is formally described in Fig. 10. As we emphasized, CDS encoding is not a MPRE. For correctness, the secret s can be decoded if $\mathbf{a}_1\mathbf{a}_1^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$. For privacy, the encoding can be simulated given the corrupted parties' input and the following information:

- The secret s if $\mathbf{a}_1\mathbf{a}_1^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$.
- $p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle$, $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$, $p_3 = \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1\mathbf{q}_2^\intercal \rangle$, where $\mathbf{q}_1 = (q_1, 1), \mathbf{q}_2 = (q_2, 1)$ are sampled by party P_3. (If the adversary corrupts P_3, it can adaptively choose q_1, q_2.)

Let us convey some intuitions why the three goals of CDS encoding in Fig. 10 are achieved.

- For disclosing s: If $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{B}_1, \mathbf{B}_2)$ is in the support of tensor-OLE correlated randomness, then $p_1p_2 = p_3$, thus the matrix $\left[\begin{smallmatrix} 1 & p_2 \\ p_1 & p_3 \end{smallmatrix}\right]$ is not full-rank. The secret s can still be recovered from $\mathbf{c}$.

- For hiding s: If $a_1a_2 \neq b_1 + b_2$, it means $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{B}_1, \mathbf{B}_2)$ is not tensor-OLE correlation. Then $p_1p_2 \neq p_3$ with high probability due to the randomness of q_1, q_2. Since matrix $\left[\begin{smallmatrix} 1 & p_2 \\ p_1 & p_3 \end{smallmatrix}\right]$ is full-rank, the secret s is perfectly masked by (r_1, r_2).
- The encoding also leaks information about $\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}_2, \mathbf{B}_2$, but as we will show, a_1, b_1, a_2, b_2 remain hidden. p_1, p_2 are one-time padded by $\mathbf{a}_1[2], \mathbf{a}_2[2]$. We leave the analysis of p_3 to the next section.
- The encoding function outputs $p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle$, $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$,

$$p_3 = \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \left[\begin{smallmatrix} q_1q_2 & q_1 \\ q_2 & 1 \end{smallmatrix}\right] \rangle, \quad \mathbf{c} = \begin{bmatrix} r_1 + \langle \mathbf{a}_2, \left[\begin{smallmatrix} r_2q_2 \\ r_2 \end{smallmatrix}\right] \rangle \\ \langle \mathbf{a}_1, \left[\begin{smallmatrix} r_1q_1 \\ r_1 \end{smallmatrix}\right] \rangle + \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \left[\begin{smallmatrix} r_2q_1q_2 & r_2q_1 \\ r_2q_2 & r_2 \end{smallmatrix}\right] \rangle \end{bmatrix}$$

  has degree 2 after $r_1q_1, r_2q_2, q_1q_2, r_2q_1, r_2q_1q_2$ are locally computed by P_3.

4.3 Semi-Malicious MPRE for Degree-3 Functions

Based on the CDS encoding (Fig. 10) we discussed in Sec. 4.2, we construct a semi-maliciously secure MPRE (Fig. 11) for canonical degree-3 functions. The MPRE has effective degree 2.

The idea is simple: Three parties P_1', P_2', P_3' need to compute $x_1x_2x_3 + z_1 + z_2 + z_3$. Every party P_i' samples a random mask s_i, one-time pads the output by s_i, and reveals s_i if and only if P_1' and P_2' use legit OLE correlated randomness. The last operation is allowed by our CDS encoding.

Lemma 4.1. *Fig. 11 presents a semi-maliciously secure MPRE for degree-3 function f, whose effective degree is 2.*

The rest of the section is going to prove this lemma.

Effective Degree. For each $t \notin \mathcal{I}$, the part of encoding corresponding to f_t is clearly of degree ≤ 2. For each $t \in \mathcal{I}$, the matrix M_t is a degree-2 function, the corresponding CDS encodings has effective degree 2, as we have analyzed in Sec. 4.2. Note that the preprocessing is very simple: Preprocessing only need to compute monomials of degree at most 3.

Efficiency Analysis. For each $t \notin \mathcal{I}$, the term f_t is preserved in $\hat{f}$. There is no complexity blow-up, and it requires no computation in the preprocessing.

For each $t \in \mathcal{I}$, the term f_t corresponds to consumption of a 4×4 OLE correlated randomness, $O(1)$ monomials in $\hat{f}$, and requires preprocessing. As we just noticed in the discussion about effective degree, the required preprocessing is very simple. The corresponding preprocessing

- consists of $O(1)$ multiplication gates;
- the outcome of the preprocessing is only used by $O(1)$ monomials of $\hat{f}$. (Because f_t corresponds to only $O(1)$ monomials in $\hat{f}$.)

These properties will improve the efficiency of the MPC protocol in Sec. 5.1.

Correctness. For each $t \notin \mathcal{I}$, f_t is correctly computed by the encoding function.

Figure 11: **The Semi-Malicious MPRE For Degree-3 Functions**

Function Input: P_i has $\mathbf{x}_i \in \mathbb{F}^{\ell_i}$

Function: A canonical degree-3 function f. By definition, there exists an index set $\mathcal{I}$. For each $t \notin \mathcal{I}$, the degree of f_t is at most 2. For each $t \in \mathcal{I}$, f_t equals $\mathbf{x}_{i_{t,1}}[j_{t,1}] \cdot \mathbf{x}_{i_{t,2}}[j_{t,2}] \cdot \mathbf{x}_{i_{t,3}}[j_{t,3}] + \mathbf{x}_{i_{t,1}}[j'_{t,1}] + \mathbf{x}_{i_{t,2}}[j'_{t,2}] + \mathbf{x}_{i_{t,3}}[j'_{t,3}]$ for some $i_{t,1}, i_{t,2}, i_{t,3} \in [n]$ and $j_{t,i}, j'_{t,i} \in [\ell_i]$.

As long as t is clear in the context, we refer to $P_{i_{t,1}}, P_{i_{t,2}}, P_{i_{t,3}}$ as P'_1, P'_2, P'_3 resp. and refer to $\mathbf{x}_{i_{t,1}}[j_{t,1}], \mathbf{x}_{i_{t,2}}[j_{t,2}], \mathbf{x}_{i_{t,3}}[j_{t,3}], \mathbf{x}_{i_{t,1}}[j'_{t,1}], \mathbf{x}_{i_{t,2}}[j'_{t,2}], \mathbf{x}_{i_{t,3}}[j'_{t,3}]$ as $x_1, x_2, x_3, z_1, z_2, z_3$ resp.

Randomness: For each $t \in \mathcal{I}$, P'_1 samples $s_{t,1}, a_{t,4,1} \in \mathbb{F}$, P'_2 samples $s_{t,2}, a_{t,5,2} \in \mathbb{F}$, P'_3 samples $s_{t,3}, a_{t,3}, a_{t,4,3}, a_{t,5,3} \in \mathbb{F}$. They also sample additional randomness according to the sub-module of CDS encoding (Fig. 10).

We will omit t in subscript, if there is no confusion.

Correlated Randomness: For each $t \in \mathcal{I}$, P'_1 receives $\mathbf{a}_{t,1} \in \mathbb{F}^4, \mathbf{B}_{t,1} \in \mathbb{F}^{4\times 4}$, P'_2 receives $\mathbf{a}_{t,2} \in \mathbb{F}^4, \mathbf{B}_{t,2} \in \mathbb{F}^{4\times 4}$ where $(\mathbf{a}_{t,1}, \mathbf{a}_{t,2}, \mathbf{B}_{t,1}, \mathbf{B}_{t,2})$ is sampled from 4×4 tensor OLE correlated randomness.

We will omit t from the subscript, if there is no confusion.

Preprocessing: Required by the sub-module of CDS encoding (Fig. 10).

Encoding Function: For each $t \notin \mathcal{I}$, output f_t.

For each $t \in \mathcal{I}$, output

$$M_t = \widehat{\mathsf{3MultPlus}}\Big(\big((x_1, z_1 + s_1), a_{4,1}, (a_1, b_1)\big), \big((x_2, z_2 + s_2), a_{5,2}, (a_2, b_2)\big), \big((x_3, z_3 + s_3), (a_3, a_{4,3}, a_{5,3}), \perp\big)\Big)$$

where $a_1 := \mathbf{a}_1[1], a_2 := \mathbf{a}_2[1], b_1 := \mathbf{B}_1[1,1], b_2 := \mathbf{B}_2[1,1]$.

P'_1, P'_2, P'_1 use CDS encoding to disclose s_1, conditioning on

$$\begin{bmatrix}\mathbf{a}_1[1]\\ \mathbf{a}_1[3]\end{bmatrix}\begin{bmatrix}\mathbf{a}_2[1]\\ \mathbf{a}_2[3]\end{bmatrix}^\intercal = \begin{bmatrix}\mathbf{B}_1[1,1] & \mathbf{B}_1[1,3]\\ \mathbf{B}_1[3,1] & \mathbf{B}_1[3,3]\end{bmatrix} + \begin{bmatrix}\mathbf{B}_2[1,1] & \mathbf{B}_2[1,3]\\ \mathbf{B}_2[3,1] & \mathbf{B}_2[3,3]\end{bmatrix}^\intercal$$

P'_1, P'_2, P'_2 use CDS encoding to disclose s_2, conditioning on

$$\begin{bmatrix}\mathbf{a}_1[1]\\ \mathbf{a}_1[4]\end{bmatrix}\begin{bmatrix}\mathbf{a}_2[1]\\ \mathbf{a}_2[4]\end{bmatrix}^\intercal = \begin{bmatrix}\mathbf{B}_1[1,1] & \mathbf{B}_1[1,4]\\ \mathbf{B}_1[4,1] & \mathbf{B}_1[4,4]\end{bmatrix} + \begin{bmatrix}\mathbf{B}_2[1,1] & \mathbf{B}_2[1,4]\\ \mathbf{B}_2[4,1] & \mathbf{B}_2[4,4]\end{bmatrix}^\intercal$$

P'_1, P'_2, P'_3 use CDS encoding to disclose s_3, conditioning on

$$\begin{bmatrix}\mathbf{a}_1[1]\\ \mathbf{a}_1[2]\end{bmatrix}\begin{bmatrix}\mathbf{a}_2[1]\\ \mathbf{a}_2[2]\end{bmatrix}^\intercal = \begin{bmatrix}\mathbf{B}_1[1,1] & \mathbf{B}_1[1,2]\\ \mathbf{B}_1[2,1] & \mathbf{B}_1[2,2]\end{bmatrix} + \begin{bmatrix}\mathbf{B}_2[1,1] & \mathbf{B}_2[1,2]\\ \mathbf{B}_2[2,1] & \mathbf{B}_2[2,2]\end{bmatrix}^\intercal$$

Decoding Function: For each $t \notin \mathcal{I}$, output f_t.

For each $t \in \mathcal{I}$, decode the CDS encoding to recover s_1, s_2, s_3. If s_1, s_2, s_3 are recovered, output $\det M_t - s_1 - s_2 - s_3$. Otherwise, output $\perp$.

For each $t \in \mathcal{I}$, since $b_1 + b_2 = a_1 a_2$ holds,

$$\begin{aligned}\det M_t &= \det \begin{bmatrix} 1 & a_1 & a_4 \\ & 1 & \\ & & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 & & \begin{pmatrix} z_1+z_2+z_3+ \\ s_1+s_2+s_3 \end{pmatrix} \\ -1 & x_3 & \\ & -1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & a_3 & a_5 \\ & 1 & a_2 \\ & & 1 \end{bmatrix} \\ &= \det \begin{bmatrix} x_1 & & \begin{pmatrix} z_1+z_2+z_3+ \\ s_1+s_2+s_3 \end{pmatrix} \\ -1 & x_3 & \\ & -1 & x_2 \end{bmatrix} \\ &= x_1x_2x_3 + z_1 + z_2 + z_3 + s_1 + s_2 + s_3. \end{aligned} \tag{5}$$

Thus $\det M - s_1 - s_2 - s_3$ equals f_t, as long as the decoding function of the CDS encoding correctly recovers s_1, s_2, s_3.

Correctness of the CDS encoding. Since $\mathbf{a}_1\mathbf{a}_2^\intercal = \mathbf{B}_1 + \mathbf{B}_2^\intercal$, then

$$p_3 = \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1\mathbf{q}_2^\intercal \rangle = \langle \mathbf{a}_1\mathbf{a}_2^\intercal, \mathbf{q}_1\mathbf{q}_2^\intercal \rangle = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle \cdot \langle \mathbf{a}_2, \mathbf{q}_2 \rangle = p_1 p_2 \ .$$

The check is passed and CDS encoding outputs

$$\begin{bmatrix} -p_1 \\ 1 \end{bmatrix}^\intercal \cdot \mathbf{c} = \underbrace{\begin{bmatrix} -p_1 \\ 1 \end{bmatrix}^\intercal \cdot \begin{bmatrix} 1 & p_2 \\ p_1 & p_3 \end{bmatrix}}_{\text{zero}} \cdot \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} + \begin{bmatrix} -p_1 \\ 1 \end{bmatrix}^\intercal \cdot \begin{bmatrix} 0 \\ s \end{bmatrix} = s \ .$$

Semi-Malicious Security. For each corrupted party $P_i \in C$, the adversary chooses the input $\mathbf{x}_i$, randomness, and correlated randomness that P_i input to the ideal functionality $\mathcal{F}^{\mathsf{os}}_{\hat{f} \circ h}$. From them, the simulator extracts $\mathbf{x}_i$ as the input of P_i in the ideal execution.

Upon receiving the output of f, the simulator need to simulate the encoding. For each $t \notin \mathcal{I}$, the corresponding part of the encoding can be trivially simulated, as it equals f_t. For each $t \in \mathcal{I}$, the simulator need to simulate matrix M_t and the encoding produced by CDS encoding, as follows:

- If P'_1 is corrupted, the simulator can compute $M_t[1,1] = x_1 - a_1$, since both x_1, a_1 are chosen by the adversary and are known to the simulator. Otherwise, simulate $M_t[1,1]$ as uniform random, because it is one-time padded by a_1. In either case, $M_t[1,1]$ can be simulated.

  Similarly $M_t[2,2]$ and $M_t[3,3]$ can be simulated.

- If both P'_1 and P'_3 are corrupted, the simulator can compute $M_t[1,2]$. Otherwise, simulate $M_t[1,2]$ as uniform random, because it is one-time padded by $a_{4,1}$ or $a_{4,3}$.

  Similarly $M_t[2,3]$ can be simulated.

- The encoding generated by the CDS encoding will be simulated by a sub-simulator, which will be described later. For each $i \in \{1,2,3\}$, the sub-simulator simulates the CDS encoding involving P'_1, P'_2, P'_i that discloses s_i conditioning on

$$\begin{bmatrix} \mathbf{a}_1[1] \\ \mathbf{a}_1[i'] \end{bmatrix} \begin{bmatrix} \mathbf{a}_2[1] \\ \mathbf{a}_2[i'] \end{bmatrix}^\intercal = \begin{bmatrix} \mathbf{B}_1[1,1] & \mathbf{B}_1[1,i'] \\ \mathbf{B}_1[i',1] & \mathbf{B}_1[i',i'] \end{bmatrix} + \begin{bmatrix} \mathbf{B}_2[1,1] & \mathbf{B}_2[1,i'] \\ \mathbf{B}_2[i',1] & \mathbf{B}_2[i',i'] \end{bmatrix}^\intercal \tag{6}$$

  where $i' := i + 1$. The sub-simulator is interactive, it asks the simulator for some additional information.

- If P'_1 (resp. P'_2) is not corrupted, the sub-simulator may choose $q \in \mathbb{F}$, and ask the simulator for $q \cdot \mathbf{a}_1[1] + \mathbf{a}_1[i']$ (resp. $q \cdot \mathbf{a}_2[1] + \mathbf{a}_2[i']$). The simulator simulates it as uniform random, since it is one-time padded by $\mathbf{a}_1[i']$ (resp. $\mathbf{a}_2[i']$).
- If P'_i is not corrupted and condition (6) is satisfied, the sub-simulator may ask the simulator for s_i. In such case, the simulator simulates s_i as uniform random. Here "condition (6) is satisfied" means either the adversary does not tamper with the correlated randomness, or the adversary corrupts both P_1, P_2 and knows (6) holds after tampering.

- The adversary may tamper with the correlated randomness.

  If after the tampering, for some honest P'_i, the condition (6) is not satisfied, the simulator can simulate $M_t[1,3]$ as uniform random, because it is one-time padded by s_i.

  Otherwise, each of s_i is revealed, either because P'_i is corrupted, or because the condition (6) is satisfied and s_i is simulated under sub-simulator's request. The condition (6) implies that $a_1 a_2 = b_1 + b_2$ after tampering. The simulator also knows $f_t = x_1 x_2 x_3 + z_1 + z_2 + z_3$. In such case the simulator can computes $M_t[1,3]$, which is uniquely determined by (5).

Sub-simulator The CDS encoding is presented in Fig. 10. Let P_1, P_2, P_3 denotes the three parties specified by the outer MPRE. We just enumerate the simulation by the set of corrupted parties. As we mentioned, the sub-simulator can choose $q_i \in \mathbb{F}$ and ask for $\langle \mathbf{a}_i, q_i \| 1 \rangle$, the sub-simulator can also ask for s if the condition is satisfied.

- The case P_1, P_2, P_3 are all corrupted: Trivial simulation.

- The case nobody is corrupted: Sample q_1, q_2 at random. Ask the outer simulator for $p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle$, $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$, where $\mathbf{q}_1, \mathbf{q}_2$ are defined as $\mathbf{q}_k = q_k \| 1$. Compute $p_3 = p_1 p_2$. Simulate $\mathbf{c}$ as uniform random, since it is one-time padded by $r_1 \| r_2$.

- The case P_3 is corrupted: q_1, q_2 are chosen by the adversary. Ask the outer simulator for $p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle$, $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$. Compute $p_3 = p_1 p_2$. Simulate $\mathbf{c}$ as uniform random, since it is one-time padded by $r_1 \| r_2$.

- The case P_1, P_3 are corrupted: q_1, q_2 are chosen by the adversary. The corrupted P_1 can replace $\mathbf{a}_1, \mathbf{B}_1$ by $\mathbf{a}'_1, \mathbf{B}'_1$. Compute $p_1 = \langle \mathbf{a}'_1, \mathbf{q}_1 \rangle$. Ask the outer simulator for $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$. Computed p_3 from

$$\begin{aligned} p_3 = \langle \mathbf{B}'_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1 \mathbf{q}_2^\intercal \rangle &= \langle \mathbf{B}_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1 \mathbf{q}_2^\intercal \rangle + \langle \mathbf{B}'_1 - \mathbf{B}_1, \mathbf{q}_1 \mathbf{q}_2^\intercal \rangle \\ &= \langle \mathbf{a}_1, \mathbf{q}_1 \rangle \cdot p_2 + \langle \mathbf{B}'_1 - \mathbf{B}_1, \mathbf{q}_1 \mathbf{q}_2^\intercal \rangle. \end{aligned} \quad (7)$$

  Note that the simulator knows $\mathbf{a}_1, \mathbf{B}_1, \mathbf{a}'_1, \mathbf{B}'_1$. Compute $\mathbf{c} = \left[\begin{smallmatrix} 1 & p_2 \\ p_1 & p_3 \end{smallmatrix}\right] \left[\begin{smallmatrix} r_1 \\ r_2 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 0 \\ s \end{smallmatrix}\right]$, since r_1, r_2, s are all chosen by the adversary.

- The case P_1, P_2 are corrupted: The corrupted P_1 can replace $\mathbf{a}_1, \mathbf{B}_1$ by $\mathbf{a}'_1, \mathbf{B}'_1$. The corrupted P_2 can replace $\mathbf{a}_2, \mathbf{B}_2$ by $\mathbf{a}'_2, \mathbf{B}'_2$.

  If $\mathbf{a}'_1 (\mathbf{a}'_2)^\intercal = \mathbf{B}'_1 + (\mathbf{B}'_2)^\intercal$, the sub-simulator can ask the outer simulator for s. Simulation becomes easy because the simulator knows all private information of P_3.

  If $\mathbf{a}'_1 (\mathbf{a}'_2)^\intercal \neq \mathbf{B}'_1 + (\mathbf{B}'_2)^\intercal$, sample q_1, q_2 at random, and compute $p_1 = \langle \mathbf{a}'_1, \mathbf{q}_1 \rangle$, $p_2 = \langle \mathbf{a}'_2, \mathbf{q}_2 \rangle$, $p_3 = \langle \mathbf{B}'_1 + (\mathbf{B}'_2)^\intercal, \mathbf{q}_1 \mathbf{q}_2^\intercal \rangle$. Then $p_1 p_2 \neq p_3$ with overwhelming probability due to the randomness of q_1, q_2. Simulate $\mathbf{c}$ as uniform random, since it is one-time padded by $r_1 \| r_2$.

- The case P_1 is corrupted: The corrupted P_1 can replace $\mathbf{a}_1, \mathbf{B}_1$ by $\mathbf{a}'_1, \mathbf{B}'_1$. Sample q_1, q_2 at random, ask the outer simulator for $p_2 = \langle \mathbf{a}_2, \mathbf{q}_2 \rangle$. Compute $p_1 = \langle \mathbf{a}_1, \mathbf{q}_1 \rangle$, and compute $p_3 = \langle \mathbf{B}'_1 + \mathbf{B}_2^\intercal, \mathbf{q}_1 \mathbf{q}_2^\intercal \rangle$ from Equation (7).

  If $(\mathbf{a}'_1, \mathbf{B}'_1) = (\mathbf{a}_1, \mathbf{B}_1)$, the sub-simulator can ask the outer simulator for s. Then sample random r_1, r_2 and compute $\mathbf{c} = \left[\begin{smallmatrix} 1 & p_2 \\ p_1 & p_3 \end{smallmatrix}\right] \left[\begin{smallmatrix} r_1 \\ r_2 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 0 \\ s \end{smallmatrix}\right]$.

  If $(\mathbf{a}'_1, \mathbf{B}'_1) \neq (\mathbf{a}_1, \mathbf{B}_1)$, then $\mathbf{a}'_1 \mathbf{a}_2^\intercal \neq \mathbf{B}'_1 + \mathbf{B}_2^\intercal$ with overwhelming probability due to the randomness of $\mathbf{a}_2$, then consequently $p_1 p_2 \neq p_3$ with overwhelming probability due to the randomness of q_1, q_2. The sub-simulator simulates $\mathbf{c}$ as uniform random, since it is one-time padded by $r_1 \| r_2$.

5 MPC Protocol for Effective-Degree-2 Functions

In this section, we construct a 2-round MPC protocol for computing an effective-degree-2 function g. The protocol securely implement $\mathcal{F}_g^{\mathsf{os}}$ against malicious corruptions. It tolerates $n-1$ corruptions, uses random oracle, and consumes tensor-OLE correlated randomness.

A protocol consuming correlated randomness can be equivalent stated as an online-offline protocol. The offline phase uses the $\mathcal{F}_{\mathsf{OLEcor}}$ (which will be implemented in Sec. A) to prepare tensor-OLE correlated randomness, and the online phase uses $\mathcal{F}_{\mathsf{RO}}$ and has only 2 rounds.

Canonical Form. Before we construct MPC protocols for effective-degree-2 functions, we observe that it is w.l.o.g. to assume the effective-degree-2 function g has the following properties.

- By definition, an effective-degree-2 function $g : \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_n} \to \mathbb{F}^{\ell}$ can be decomposed into a degree-2 function $\hat{f} : \mathbb{F}^{\hat{\ell}_1} \times \cdots \times \mathbb{F}^{\hat{\ell}_n} \to \mathbb{F}^{\ell}$ and preprocessing functions $h_1 : \mathbb{F}^{\ell_1} \to \mathbb{F}^{\hat{\ell}_1}, \ldots, h_n : \mathbb{F}^{\ell_n} \to \mathbb{F}^{\hat{\ell}_n}$ such that $g := f \circ h$.

- We assume $\hat{f}$ is a **canonical degree-2 function**. That is, there is there is an index set $\mathcal{I} \subseteq [\ell]$, such that for each $t \in [\ell]$,

  - If $t \notin \mathcal{I}$, the t-th coordinate of f, denoted by f_t, is a linear function.
  - If $t \in \mathcal{I}$,
  $$f_t(\mathbf{x}_1, \ldots, \mathbf{x}_n) := \mathbf{x}_{i_t}[\iota_1] \cdot \mathbf{x}_{j_t}[\iota_2] + \mathbf{x}_{i_t}[\iota_3] + \mathbf{x}_{j_t}[\iota_4]$$
  for some $i_t, j_t \in [n]$ and $\iota_1, \iota_2 \in [\ell_{i_t}]$, $\iota_3, \iota_4 \in [\ell_{j_t}]$.
  As long as t is clear from the context, we refer to $P_{i_t}, \mathbf{x}_{i_t}[\iota_1], \mathbf{x}_{j_t}[\iota_2], P_{j_t}, \mathbf{x}_{i_t}[\iota_3], \mathbf{x}_{j_t}[\iota_4]$ as $P_i, v_i, z_i, P_j, v_j, z_j$, respectively. With such aliases, $f_t := v_i v_j + z_i + z_j$.

  Similar to our discussion about canonical degree-3 functions in Sec. 4, assuming $\hat{f}$ has canonical form will increase the monomial complexity of $\hat{f}$ by at most a constant factor.

- We assume that the circuit computing h_i only has multiplication gates, because the preprocessing function of our effective-degree-2 MPRE only has multiplication gates.

  We assume that h_i output its input in the first ℓ_t output coordinates, each of the later output coordinate is the product of two previous output coordinates. For each $t \leq \ell_i$, $h_i(\mathbf{x}_i)[t] := \mathbf{x}_i[t]$. For each $\ell_i < t \leq \hat{\ell}_i$, there exists some $t_1, t_2 < t$ such that
  $$h_i(\mathbf{x}_i)[t] := h_i(\mathbf{x}_i)[t_1] \cdot h_i(\mathbf{x}_i)[t_2] \,.$$
  Our assumptions on h_i will not increase its circuit size.

Remark: It is easy to extend our construction to cover general arithmetic preprocessing functions. Proving addition is even simpler than multiplication.

Protocol Computing 2MultPlus. Define function 2MultPlus to capture the degree-2 terms in canonical degree-2 functions.

$$\mathsf{2MultPlus} : ((v_1, z_1), (v_2, z_2)) \mapsto v_1 v_2 + z_1 + z_2.$$

Lin et al. [LLW20] present a semi-honest 2-round MPC protocol $\Pi_{\mathsf{2MultPlus}}$ computing 2MultPlus (Fig. 12), using (scalar) OLE correlated randomness.

Figure 12: MPC Protocol $\Pi_{\mathsf{2MultPlus}}$

Input: Party P_1 has $v_1, z_1 \in \mathbb{F}$. Party P_2 has $v_2, z_2 \in \mathbb{F}$.

Correlated Randomness: Sample $a_1, a_2, b_1, b_2 \in \mathbb{F}$ such that $a_1 a_2 = b_1 + b_2$. Party P_1 receives a_1, b_1. Party P_2 receives a_2, b_2.

Round 1: Party P_1 broadcasts $c_1 := v_1 + a_1$. Party P_2 broadcasts $c_2 := v_2 + a_2$.

Round 2: Party P_1 broadcasts $m_1 := v_1 c_2 + b_1 + z_1$. Party P_2 broadcasts $m_2 := v_2 c_1 + b_2 + z_2$.

Output: Output $m_1 + m_2 - c_1 c_2$.

Note that in an honest execution,

$$\begin{aligned} m_1 + m_2 - c_1 c_2 &= \underset{\uparrow}{v_1 c_2} + b_1 + z_1 + \underset{\uparrow}{v_2 c_1} + b_2 + z_2 - \underset{\uparrow}{c_1 c_2} \\ &\qquad\quad v_2 + a_2 \qquad\qquad\quad v_1 + a_1 \qquad\quad (v_1 + a_1)(v_2 + a_2) \\ &= v_1 v_2 + z_1 + z_2 + b_1 + b_2 - a_1 a_2 \\ &= v_1 v_2 + z_1 + z_2 \,. \end{aligned}$$

It is not hard to show that protocol $\Pi_{\mathsf{2MultPlus}}$ is actually maliciously secure with output substitution. Thus a natural approach is to use parallel sessions of the protocol to compute all degree-2 terms in g. Each session of $\Pi_{\mathsf{2MultPlus}}$ computes one coordinate of $\hat{f}$. This natural approach faces two challenges:

Consistency. Say two coordinates of $\hat{f}$ equal $v_i v_j + z_i + z_j$ and $v_i v_k + z'_i + z'_k$ respectively. We have to ensure that P_i feeds the *same* v_i to the two corresponding sessions.

Well-Formedness. We have to ensure that P_i evaluates the local preprocessing function $\hat{\mathbf{x}}_i = h_i(\mathbf{x}_i)$ correctly.

Inspired by the protocol $\Pi_{\mathsf{2MultPlus}}$ and the security concerns of the natural approach, we construct a maliciously secure protocol (Fig. 14) for computing effective-degree-2 functions. To formalize the construction, we define the enhanced 2MultPlus functionality $\mathcal{F}_{\mathsf{2MP^+}}$ (Fig. 13), which offers four commands:

Input command captures the first round of $\Pi_{\mathsf{2MultPlus}}$.

2MultPlus command captures the second round of $\Pi_{\mathsf{2MultPlus}}$.

ProveSame command deals with the consistency concern. Via this command, a party can prove that he feeds the same value in two different sessions.

ProveProd command deals with the well-formedness concern. Via this command, a party can prove that the value he fed in a session equals the product of the values he fed in two other sessions. (Recall that we assume the preprocessing circuit only has multiplication gates.)

Figure 13: The enhanced 2MultPlus functionality $\mathcal{F}_{\mathsf{2MP}^+}$.

The functionality is initialized with a field $\mathbb{F}$ and n parties $P_1, \cdots, P_n$, where a subset $C \subset [n]$ of parties are corrupted by an adversary. All variables and computations below are in $\mathbb{F}$, unless specified otherwise.

The functionality offers 4 kinds of command, Input, ProveProd, ProveSame and 2MultPlus. Input command and 2MultPlus command take a session id $\mathsf{sid} = (\mathsf{2MP}, \{i, j\}, \mathsf{seqnum})$ as a part of the input. In the session, P_i, P_j jointly computes the 2MultPlus function. ProveProd command takes a session id $\mathsf{sid} = (\mathsf{Pf}, i, j, \mathsf{seqnum})$ as a part of the input. In the session, P_i proves an argument to P_j.

These commands must be executed in a given order. A party can invoke Input or ProveProd command only before it invokes any ProveSame command. A party can invoke 2MultPlus command only after both parties in the session have invoked Input command.

Input: Party P_i commits to its input in session $\mathsf{sid} = (\mathsf{2MP}, \{i, j\}, \mathsf{seqnum})$.

On input $(\texttt{"Input"}, \mathsf{sid}, (\mathrm{var}, v))$ from P_i, where var is a variable name, and $v \in \mathbb{F}$. The functionality

- Stores $(\mathsf{sid}, P_i, (\mathrm{var}, v))$
- Broadcasts $(\mathsf{sid}, P_i, \mathrm{var}, \texttt{"Received"})$ to all parties.

and ignore future Input command with same sid from P_i.

ProveProd: Party P_i proves to P_j that var equals the product of var_1 and var_2, in session $\mathsf{sid} = (\mathsf{Pf}, i, j, \mathsf{seqnum})$.

On input $(\texttt{"ProveProd"}, \mathsf{sid}, (\mathrm{var}_1, v_1), (\mathrm{var}_2, v_2), (\mathrm{var}_3, v_3))$ from P_i, if $P_i \in C$, the functionality additionally receives $1_{\mathsf{rej}} \in \{0, 1\}$ from the adversary. The functionality

- Stores $(\mathsf{sid}, P_i, (\mathrm{var}_1, v_1)), (\mathsf{sid}, P_i, (\mathrm{var}_2, v_2)), (\mathsf{sid}, P_i, (\mathrm{var}_3, v_3))$.
- Broadcasts $(\mathsf{sid}, P_i, \mathrm{var}, \mathrm{var}_2, \mathrm{var}_3, \texttt{"Received"})$ to all parties.
- Sends $(\mathsf{sid}, P_i, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathsf{acc})$ to the other participant P_j, if $v_1 v_2 = v_3$ and $1_{\mathsf{rej}} = 0$. Otherwise sends $(\mathsf{sid}, P_i, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathsf{rej})$ to P_j.

and ignore future ProveProd command with same sid from P_i.

ProveSame: On input $(\texttt{"ProveSame"}, (\mathrm{var}, u))$ from P_i. If $P_i \in C$, the functionality asked the adversary to additionally provides an indicator vector $\mathbf{1}_{\mathsf{rej}} \in \{0, 1\}^n$.

For every party P_j, retrieve all entries $(\mathsf{sid}_\ell, P_i, (\mathrm{var}, v_\ell))$ corresponding to sessions between P_i, P_j (i.e., sid_ℓ contains P_i, P_j). Send $(P_i, \mathrm{var}, \mathsf{rej})$ to P_j if there exists $v_\ell \neq u$ or if $P_i \in C$ and $\mathbf{1}_{\mathsf{rej}}[j] = 1$. Otherwise, send $(P, \mathrm{var}, \mathsf{acc})$ to P_j.

The functionality ignores further "ProveSame" messages for var from P.

This means P proves that it used the same value for var *in different sessions. All parties receiving* acc *are guaranteed to have received commitments to the same value.*

2MultPlus: On input $(\texttt{"2MultPlus"}, \mathsf{sid} = (\mathsf{2MP}, \{i, j\}, \mathsf{seqnum}), z_i)$ from P_i and input $(\texttt{"2MultPlus"}, \mathsf{sid}, z_j)$ from P_j, the functionality retrieves $(\mathsf{sid}, P_i, (\star, v_i))$ and $(\mathsf{sid}, P_j, (\star, v_j))$ (ignore if no such entries exist), computes $y = v_i v_j + z_i + z_j$, and sends $(\texttt{"Output"}, \mathsf{sid}, y)$ to *the adversary*.

Then, on input $(\texttt{"Output"}, \mathsf{sid}, y')$ from the adversary, the functionality broadcasts $(\texttt{"Output"}, \mathsf{sid}, y')$ to all parties. A passive adversary always lets $y' = y$.

This means P_i and P_j can compute 2MultPlus *using inputs v_i, v_j that they committed to in the same session* sid.

Figure 14: Protocol Computing Effective-Degree-2 Functions

First Round Party P_i locally preprocesses $\hat{\mathbf{x}}_i = h_i(\mathbf{x}_i)$. Let "X–$i$–$t$" be the variable name of $\hat{\mathbf{x}}_i[t]$. Party P_i then uses the commands in $\mathcal{F}_{\mathsf{2MP}^+}$ to

Commit to Inputs: For every $j \neq i$ and for each output coordinate of $\hat{f}$ that is of the form $\hat{\mathbf{x}}_i[t]\hat{\mathbf{x}}_j[t''] + \hat{\mathbf{x}}_i[t'] + \hat{\mathbf{x}}_j[t''']$. Let $\mathsf{sid} = (\mathsf{2MP}, \{i, j\}, \mathsf{seqnum})$ be the session id for computing this coordinate. Party P_i calls

$$\Big(\texttt{"Input"}, \mathsf{sid}, (\texttt{"X–}i\texttt{–}t\texttt{"}, \hat{\mathbf{x}}_i[t])\Big)$$

to commit to $\hat{\mathbf{x}}_i[t]$ under this session.

Prove Well-Formedness: For every $j \neq i$, P_i prove to P_j that $\hat{\mathbf{x}}_i$ is honestly computed, by calling the following for each $t \in \mathcal{J}_{i,j}$ that $t \geq \ell_i$,

$$\Big(\texttt{"ProveProd"}, (\mathsf{Pf}, i, j, t), (\texttt{"X–}i\texttt{–}t_1\texttt{"}, \hat{\mathbf{x}}_i[t_1]), (\texttt{"X–}i\texttt{–}t_2\texttt{"}, \hat{\mathbf{x}}_i[t_2]), (\texttt{"X–}i\texttt{–}t\texttt{"}, \hat{\mathbf{x}}_i[t])\Big)$$

where $t_1, t_2 < t$ are the corresponding indexes that $\hat{\mathbf{x}}[t] := \hat{\mathbf{x}}[t_1] \cdot \hat{\mathbf{x}}[t_2]$.

Prove Consistency: For each $t \leq \hat{\ell}_i$, P_i calls

$$\Big(\texttt{"ProveSame"}, (\texttt{"X–}i\texttt{–}t\texttt{"}, \hat{\mathbf{x}}_i[t])\Big).$$

Second Round If party P_i receives any rej from $\mathcal{F}_{\mathsf{2MP}^+}$, indicating an invalid proof, P_i will broadcast $\perp$ and abort.

If all proofs are accepted, P_i proceeds with the computation of $\hat{f}$. For every $j \neq i$ and for each output coordinate of $\hat{f}$ that is of the form $\hat{\mathbf{x}}_i[t]\hat{\mathbf{x}}_j[t''] + \hat{\mathbf{x}}_i[t'] + \hat{\mathbf{x}}_j[t''']$. Let $\mathsf{sid} = (\mathsf{2MP}, \{i, j\}, \mathsf{seqnum})$ be the session id for computing this coordinate. P_i calls

$$\Big(\texttt{"2MultPlus"}, \mathsf{sid}, (\texttt{"X–}i\texttt{–}t'\texttt{"}, \hat{\mathbf{x}}_i[t'])\Big),$$

which let $\mathcal{F}_{\mathsf{2MP}^+}$ broadcast the outcome to all parties.

Output Party P_i aborts if any party has broadcasted $\perp$.

Otherwise, P_i outputs the outcomes from "2MultPlus" command. Namely, for each coordinate of $\hat{f}$, there is a corresponding session id sid. Upon receiving $(\texttt{"Output"}, \mathsf{sid}, y)$ from $\mathcal{F}_{\mathsf{2MP}^+}$, P_i sets y be that coordinate of the output.

5.1 The Functionality $\mathcal{F}_{\mathsf{2MP}^+}$ Suffices for Effective-Degree-2 Function Evaluation

Lemma 5.1. *For any effective-degree-2 function g, there is a 2-round MPC protocol that securely implements $\mathcal{F}_g^{\mathsf{os}}$ against malicious corruptions. The protocol uses random oracle, and tensor-OLE correlated randomness.*

Proof Overview. The 2-round protocol computing g is built upon the protocol Π_{2MP^+} in Sec. 5.2 that implements $\mathcal{F}_{\mathsf{2MP}^+}$. Roughly speaking, the protocol works as follows.

First Round On input $\mathbf{x}_i$, party P_i locally computes $\hat{\mathbf{x}}_i = h_i(\mathbf{x}_i)$.

For each output coordinate of $\hat{f}$ that is of the form $v_i v_j + z_i + z_j$, party P_i commits to v_i (which is an coordinate of $\hat{\mathbf{x}}_i$) using Input command.

P_i uses ProveProd, ProveSame command to convince others that i) $\hat{\mathbf{x}}_i$ is well-formed, i.e., $\hat{\mathbf{x}}_i = h_i(\mathbf{x}_i)$; ii) the values it commits to each Input command are consistent w.r.t. $\hat{\mathbf{x}}_i$.

Second Round Party P_i receives and verifies the proofs from others. If P_i rejects a proof, it will

broadcast $\perp$ and abort[7].

Otherwise if P_i accepts all proofs, it uses 2MultPlus command to compute and broadcast each output coordinate of $\hat{f}$.

Output P_i aborts, if any party broadcasts $\perp$. Otherwise, output the outputs of the "`2MultPlus`" commands (one per output coordinate of g).

It is easy to verify correctness when all parties act honestly.

The malicious security entails two properties: (1) The adversary should not learn extra information beyond the output of g. (2) All honest parties obtain the same output, which may be dictated by the adversary.

The second property follows from public output reconstruction. Every honest party derives its output depending only on messages sent via the broadcast channel.

The first property because if the adversary violates well-formedness or consistency, the honest parties will abort since they detect it from ProveProd or ProveSame command. And if the adversary obeys well-formedness and consistency, it will only learn the output of g.

Proof. By definition and by our w.l.o.g. assumptions on $g : \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_n} \to \mathbb{F}^{\ell}$ there exist $\hat{f}, h_1, \ldots, h_n$, such that

- $g := f \circ h$. That is, $g(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \hat{f}(h_1(\mathbf{x}_1), \ldots, h_n(\mathbf{x}_n))$ for any $\mathbf{x}_1, \ldots, \mathbf{x}_n$.
- $\hat{f} : \mathbb{F}^{\hat{\ell}_1} \times \cdots \times \mathbb{F}^{\hat{\ell}_n} \to \mathbb{F}^{\ell}$ is a degree-2 arithmetic function over the field $\mathbb{F}$. Each output coordinate of $\hat{f}$ is of the canonical form

$$v_i v_j + z_i + z_j \,,$$

  where v_i, z_i come from $h_i(\mathbf{x}_i)$, v_j, z_j come from $h_j(\mathbf{x}_j)$, for some i, j.

- $h_i : \mathbb{F}^{\ell_i} \to \mathbb{F}^{\hat{\ell}_i}$ is an arithmetic circuit over $\mathbb{F}$. For each $t \le \hat{\ell}_i$,

$$h_i(\mathbf{x}_i)[t] := \begin{cases} \mathbf{x}_i[t], & \text{if } t \le \ell_i \\ h_i(\mathbf{x}_i)[t_1] \cdot h_i(\mathbf{x}_i)[t_2], & \text{if } t > \ell_i, \text{ for some } t_1, t_2 < t \end{cases}$$

For efficiency optimization, we define the sub-circuit of h_i which P_j "cares about". Party P_j want P_i to prove the t-th coordinate $h_i(\mathbf{x})[t]$ is honestly computed only if $h_i(\mathbf{x})[t]$ is used in 2MultPlus session between P_i and P_j. Denote the sub-circuit computing these coordinates of $h_i(\mathbf{x})$ by $\mathcal{J}_{i,j}$, which is a subset of wires of the preprocessing function h_i. Formally, $\mathcal{J}_{i,j}$ is recursively defined as

- For each output coordinate of $\hat{f}$ between P_i and P_j of form $v_i v_j + z_i + z_j$, say $v_i := h_i(\mathbf{x}_i)[t]$, then $t \in \mathcal{J}_{i,j}$.
- For each $t \in \mathcal{J}_{i,j}$ that $t > \ell_i$, say $h_i(\mathbf{x}_i)[t] := h_i(\mathbf{x}_i)[t_1] \cdot h_i(\mathbf{x}_i)[t_2]$, then $t_1, t_2 \in \mathcal{J}_{i,j}$.

As we have discussed in the overview, the correctness is easy to verify. The security proof is simplified by the fact that every honest party's output is determined by broadcasted messages.

The simulator works as follows: The corrupted parties invoke the commands of $\mathcal{F}_{\mathsf{2MP}^+}$ many times in the first round. In the ideal world, all these calls are captured by the simulator.

[7] Technically, "abort" means the party stops sending messages, and outputs a default value in the end. Thus if all honest parties unanimously abort, they obtain the same output.

- For each corrupted party P_i, the simulator first focuses on the "ProveSame" commands. Define vector $\hat{\mathbf{x}}_i$ by collecting values from all calls to "ProveSame". Let $\mathbf{x}_i := \hat{\mathbf{x}}_i[1{:}\ell_i]$ be the first ℓ_i coordinates of $\hat{\mathbf{x}}_i$. The simulator feeds $\mathbf{x}_i$ to the ideal functionality $\mathcal{F}_g^{\mathsf{os}}$. That is, $\mathbf{x}_i$ is the extracted input of P_i.
- The honest parties will only prove valid statements in the first round. Thus every corresponding outcome from $\mathcal{F}_{\mathsf{2MP}^+}$ is always acc, which is easily simulatable.
- The simulator can predict which honest party will broadcast $\perp$. An honest party P_j will not broadcast $\perp$ if and only if
  - For every corrupted party P_i, $\hat{\mathbf{x}}_i$ agrees with $h_i(\mathbf{x}_i)$ on all coordinates in $\mathcal{J}_{i,j}$.
  - Every corrupted party P_i, in every session between P_i and P_j, always uses consistent mapping from variable names to values. In particular, whenever P_i uses "Input" command, if the variable name is "X-i-t" then the value must be $\hat{\mathbf{x}}_i[t]$. (Combining with the previous condition, this value must equal to $h_i(\mathbf{x}_i)[t]$.)
  - P_i never turns on any abort bit that is relevant to P_j.

  In short, if and only if all corrupted parties have behaved honestly in the interaction with P_j.

Since the simulator has feed an extracted input for each corrupted party, it receives output $\mathbf{y}$ from the ideal functionality $\mathcal{F}_g^{\mathsf{os}}$.

- The simulator has identified which honest party is going to broadcast $\perp$. Simulation regarding these parties requires no extra work.
- For the rest of uncorrupted parties, thanks to the corrupted parties, who have behaved (mostly) honestly in the first round, simulation is possible given $\mathbf{y}$.

  Concretely, say the s-th coordinate of $\hat{f}$ is supposed to be $\hat{\mathbf{x}}_i[t]\hat{\mathbf{x}}_j[t''] + \hat{\mathbf{x}}_i[t'] + \hat{\mathbf{x}}_j[t''']$, party P_i is corrupted, P_j is uncorrupted. If P_j did not abort in the first round, party P_i must have behaved honestly so far, namely, P_i computes $\hat{\mathbf{x}}_i[t] = h_i(\mathbf{x}_i)[t]$ and feeds $\hat{\mathbf{x}}_i[t]$ in the first round.

  The corrupted P_i may choose an arbitrary z^* in the second round by feeding

  $$\big(\texttt{"2MultPlus"}, \mathsf{sid}, z^*\big)$$

  where sid is the corresponding session id. In the real world, the adversary will receive

  $$\big(\texttt{"Output"}, \mathsf{sid}, \hat{\mathbf{x}}_i[t]\hat{\mathbf{x}}_j[t''] + z^* + \hat{\mathbf{x}}_j[t''']\big)$$

  from $\mathcal{F}_{\mathsf{2MP}^+}$. In the ideal world, the simulator knows $\mathbf{y}[s] = \hat{\mathbf{x}}_i[t]\hat{\mathbf{x}}_j[t''] + \hat{\mathbf{x}}_i[t'] + \hat{\mathbf{x}}_j[t''']$, thus it can perfectly simulate the outcome by sending

  $$\big(\texttt{"Output"}, \mathsf{sid}, \mathbf{y}[s] - \hat{\mathbf{x}}[t'] + z^*\big)$$

  to the adversary.

In the real world, the output of every honest party is determined by the broadcasted messages. Thus the simulator can predict the output of honest parties in the real world, then feed this predicted output to $\mathcal{F}_g^{\mathsf{os}}$ as the substitution output. □

Efficiency Analysis. $\mathcal{F}_{\mathsf{2MP}^+}$ is securely implemented by the protocol in Sec. 5.2, in which

- Each Input command consumes a 3×3 OLE correlated randomness, and broadcasts $O(1)$ field elements.
- Each ProveProd command consumes a 1×11 OLE correlated randomness, and broadcasts $O(1)$ field elements.
- Each ProveSame command broadcasts $O(1)$ field elements, and sends $O(1)$ field elements per each relevant Input/ProveProd command. Notice that the later can be absorbed by the communication complexity of Input/ProveProd.
- Each MultPlus2 command broadcasts $O(1)$ field elements.

The MPC protocol in this section is built upon $\mathcal{F}_{\mathsf{2MP}^+}$.

- Each (degree-2) monomial in $\hat{f}$ requires $O(1)$ Input/MultPlus2 commands.
- Each gate in the preprocessing function requires the owner party to call ProveProd command at most n times.
- Each wire (i.e. gate or input wire) in the preprocessing function requires a ProveSame command.

Thus, if we let $\mathsf{mc}(\hat{f})$ denote the number of monomials in $\hat{f}$, let $\mathsf{cs}(h_i)$ denote the circuit size (i.e., input length plus the number of gates) of h_i, the MPC protocol for effective-degree-2 functions in this section consumes $O(\mathsf{mc}(\hat{f}) + n \cdot \sum_i \mathsf{cs}(h_i))$ instances of OLE correlated randomness, sends $O(n \cdot \sum_i \mathsf{cs}(h_i))$ field elements and broadcasts $O(\mathsf{mc}(\hat{f}) + n \cdot \sum_i \mathsf{cs}(h_i))$ field elements.

Additional efficiency improvement follows from a nice property of the effective-degree-2 MPRE in Sec. 4.3. The only computation need in preprocessing is to multiplying three inputs together, and the outcome is only used by $O(1)$ monomials in $\hat{f}$. Therefore, when the MPC protocol in this section is computing the MPRE from Sec. 4.3, the efficiency analysis can be improved to

- Each gate in the preprocessing function requires the owner party to call ProveProd command at most $O(1)$ times. (Because the owner party need to prove the gate is correctly evaluated to only $O(1)$ parties.)

The MPC protocol uses $O(\mathsf{mc}(\hat{f}) + \sum_i \mathsf{cs}(h_i))$ instances of OLE correlated randomness, sends $O(\sum_i \mathsf{cs}(h_i))$ field elements in P2P channels and broadcasts $O(\mathsf{mc}(\hat{f}) + \sum_i \mathsf{cs}(h_i))$ field elements.

Statistical Security Loss. As shown in Sec. 5.2, the total security loss is $O(\frac{\mathsf{mc}(\hat{f})+\sum_i \mathsf{cs}(h_i)+n^2 \cdot Q_{\mathsf{RO}}}{|\mathbb{F}|})$, where Q_{RO} is the number of random oracle queries that the adversary makes.

5.2 The Protocol Π_{2MP^+} Implementing $\mathcal{F}_{\mathsf{2MP}^+}$

We now describe a protocol Π_{2MP^+} for implementing $\mathcal{F}_{\mathsf{2MP}^+}$ using tensor-OLE correlated randomness and a random oracle. We will describe the protocol in pieces, starting with the offline phase setting up the correlated randomness and then moving on to sub-protocols in the online phase, each sub-protocol implements one command in $\mathcal{F}_{\mathsf{2MP}^+}$.

We assume that every honest party only execute the sub-protocols in a given order: All instances of Input/ProveProd sub-protocol first, then ProveSame, then 2MultPlus. This matches the order in $\mathcal{F}_{\mathsf{2MP}^+}$. The *Exceptional Cases* in the sub-protocol description is for enforcing order.

The Protocol Π_{2MP^+}, Offline Phase

Initialization: All parties are initialized with a finite field $\mathbb{F}$ and the number of parties n. All variables and computations below are in $\mathbb{F}$, unless specified otherwise.

Offline Phase: Every pair of parties P_i, P_j, for every computation session $\mathsf{sid} = (\mathsf{2MP}, \{i,j\}, \mathsf{seqnum})$ they will participate in together, jointly query $\mathcal{F}_{\mathsf{OLEcor}}$ with $(\texttt{"Gen"}, (i,j,3,3,\mathsf{seqnum}))$ to obtain 3×3 OLE correlated randomness. Party P_i gets random $\mathbf{a}_i \in \mathbb{F}^3$, $\mathbf{B}_i \in \mathbb{F}^{3\times 3}$, party P_j gets random $\mathbf{a}_j \in \mathbb{F}^3$, $\mathbf{B}_j \in \mathbb{F}^{3\times 3}$, such that, $\mathbf{a}_i \mathbf{a}_j^\intercal = \mathbf{B}_i + \mathbf{B}_j^\intercal$.

Every pair of parties P_i, P_j, for every proof session $\mathsf{sid} = (\mathsf{Pf}, i, j, \mathsf{seqnum})$ they will participate in together, jointly query $\mathcal{F}_{\mathsf{OLEcor}}$ with $(\texttt{"Gen"}, (i,j,11,1,\mathsf{seqnum}))$ to obtain 11×1 OLE correlated randomness. Party P_i gets random $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{F}^{11}$, party P_j gets random $a_j \in \mathbb{F}$, $\mathbf{b}_j \in \mathbb{F}^{11}$, such that, $\mathbf{a}_i \cdot a_j = \mathbf{b}_i + \mathbf{b}_j$.

This can be done in the offline phase, as long as P_i, P_j know an upper bound on the number of sessions they will participate in.

Initialize Data Structure: Each party P_i prepares the following data structure for the online phase:

- Set $S_{i,\mathsf{session}}$ will contain information for every computation session sid P_i participates in, including the OLE correlated randomness $(\mathbf{a}, \mathbf{B})$, the input variable var and value v, that is, $S_{i,\mathsf{session}} = \{(\mathsf{sid}, (\mathbf{a}, \mathbf{B}), (\mathrm{var}, v))\}_{\mathsf{sid}}$. Initially, only $\mathbf{a}, \mathbf{B}$ are set, other values are uninitialized.
- Set $S_{i,\mathsf{prover}}$ will contain information for every proof session sid P_i participates in as the prover, including the OLE correlated randomness $(\mathbf{a}, \mathbf{b})$, the input variables $\mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3$ and their values v_1, v_2, v_3, that is, $S_{i,\mathsf{prover}} = \{(\mathsf{sid}, (\mathbf{a}, \mathbf{b}), (\mathrm{var}_1, v_1), (\mathrm{var}_2, v_2), (\mathrm{var}_3, v_3))\}_{\mathsf{sid}}$. Initially, only $\mathbf{a}, \mathbf{b}$ are set, other values are uninitialized.
- Set $S_{i,\mathsf{verifier}}$ will contain information for every proof session sid P_i participates in as the verifier, which is the OLE correlated randomness $(a, \mathbf{b})$, that is, $S_{i,\mathsf{verifier}} = \{(\mathsf{sid}, (a, \mathbf{b}))\}_{\mathsf{sid}}$.
- Set $S_{i,\mathsf{guard}}$ is initialized empty and will store, for every variable var that P_i uses, its value v and a random scalar $g_{i,\mathrm{var}}$, that is $S_{i,\mathsf{guard}} = \{(\mathrm{var}, g_{i,\mathrm{var}})\}_{\mathrm{var}}$.
- Set $S_{i,\mathsf{sent}}$ is initialized empty and will store, all messages that P_i broadcasts in the online phase.

The Protocol Π_{2MP^+}, Online Phase, Input Commands

Party P_i, on $(\texttt{"Input"}, \mathsf{sid} = (\mathsf{2MP}, \{i,j\}, \mathsf{seqnum}), (\mathrm{var}, v))$ does the following.

Exceptional Cases: P_i examines the set $S_{i,\mathsf{sent}}$ of messages it has broadcast so far, and ignores the input if

1. a message of form $(\texttt{"Input"}, \mathsf{sid}, \star, \star)$ has been broadcast, or
2. a message of form $(\texttt{"ProveSame"}, \mathrm{var}, \star, \star)$ has been broadcast.

Else, P_i records (var, v) for sid in set $S_{i,\mathsf{session}}$, and proceeds to the next steps.

To commit to value v of variable var, P_i does the following:

- Retrieve $\mathbf{a}$ from $S_{i,\mathsf{session}}$ using sid.
- Find entry $(\mathrm{var}, g_{i,\mathrm{var}})$ in set $S_{i,\mathsf{guard}}$. If no such entry is found, sample $g_{i,\mathrm{var}} \leftarrow \mathbb{F}$ and add $(\mathrm{var}, g_{i,\mathrm{var}})$ to the set. $g_{i,\mathrm{var}}$ is referred to as the *guard* for var.
- To commit to v, P_i sends an OTP ciphertext of $(v \| g_{i,\mathrm{var}})$ using $\mathbf{a}[1{:}2]$ as the pad, that is,

$$\text{broadcast } \Big(\texttt{"Input"},\ \mathsf{sid},\ \mathrm{var},\ \mathbf{c} = (v + \mathbf{a}[1],\ g_{i,\mathrm{var}} + \mathbf{a}[2])\Big).$$

The Protocol $\Pi_{\mathsf{2MP^+}}$, Online Phase, ProveProd Commands

Party P_i, on ("`ProveProd`", $\mathsf{sid} = (\mathsf{Pf}, i, j, \mathsf{seqnum}), (\mathrm{var}_1, v_1), (\mathrm{var}_2, v_2), (\mathrm{var}_3, v_3)$) does the following.

Exceptional Cases: P_i examines the set $S_{i,\mathsf{sent}}$ of messages it has broadcast so far, ignores the input if

1. a message of form ("`ProveProd`", $\mathsf{sid}, \star, \star, \star$) has been broadcast, or
2. a message of form ("`ProveSame`", $X, \star, \star$) has been broadcast, for any $X \in \{\mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3\}$.

Else, P_i proceeds to the next steps.

Prover's Message If $v_1 v_2 \neq v_3$, send

$$\text{to } P_j \left(\texttt{"ProveProd"},\ \mathsf{sid},\ \mathrm{var}_1,\ \mathrm{var}_2,\ \mathrm{var}_3,\ \bot\right).$$

Otherwise if $v_1 v_2 = v_3$, the prover P_i does the following:

- Find entry $(X, g_{i,X})$ in set $S_{i,\mathsf{guard}}$ for each $X \in \{\mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3\}$. If no such entry is found, sample $g_{i,X} \leftarrow \mathbb{F}$ and add $(X, g_{i,X})$ to the set.
- Retrieve $\mathbf{a}, \mathbf{b}$ from $S_{i,\mathsf{prover}}$ using sid. Sample $g_1, g_2 \leftarrow \mathbb{F}$ and compute an OTP ciphertext
$$\mathbf{c} := (v_1 \| g_{i,\mathrm{var}_1} \| v_2 \| g_{i,\mathrm{var}_2} \| v_3 \| g_{i,\mathrm{var}_3} \| g_1 \| g_2 \| v_1 g_2 \| g_1 v_2 \| g_1 g_2) + \mathbf{a}.$$
- Query the random oracle with the generated OTP, $(q_1, q_2) \leftarrow \mathsf{RO}(\mathsf{sid}, \mathbf{c})$. q_1, q_2 determine three linear functions, represented by $q_1\|1$, $q_2\|1$ and $q_1q_2\|q_1\|q_2\|1$. Compute $p_1 = \langle q_1\|1, v_1\|g_1\rangle$, $p_2 = \langle q_2\|1, v_2\|g_2\rangle$, prepare
$$\pi_1 = \langle q_1\|1, \mathbf{b}[1]\|\mathbf{b}[7]\rangle, \quad \pi_2 = \langle q_2\|1, \mathbf{b}[3]\|\mathbf{b}[8]\rangle, \quad \pi_3 = \langle q_1q_2\|q_1\|q_2\|1, \mathbf{b}[5]\|\mathbf{b}[9{:}11]\rangle$$
as the proofs of $p_1 = \langle q_1\|1, v_1\|g_1\rangle$, $p_2 = \langle q_2\|1, v_2\|g_2\rangle$, $p_1p_2 = \langle q_1q_2\|q_1\|q_2\|1, v_3\|v_1g_2\|g_1v_2\|g_1g_2\rangle$ respectively, and
$$\text{broadcast} \left(\texttt{"ProveProd"},\ \mathsf{sid},\ \mathrm{var}_1,\ \mathrm{var}_2,\ \mathrm{var}_3,\ \mathbf{c},\ q_1,\ q_2,\ p_1,\ p_2,\ \pi_1,\ \pi_2,\ \pi_3\right).$$

Verifier's Decision: P_j decides to accept or reject as follows:

- Retrieve $a', \mathbf{b}'$ from $S_{i,\mathsf{verifier}}$ using sid.
- Reject upon receiving ("`ProveProd`", $\mathsf{sid}, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \bot$).
- Upon receiving ("`ProveProd`", $\mathsf{sid}, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{c}, q_1, q_2, p_1, p_2, \pi_1, \pi_2, \pi_3$), verify that q_1, q_2 are evaluated correctly as $(q_1, q_2) \leftarrow \mathsf{RO}(\mathsf{sid}, \mathbf{c})$. Check if
$$\pi_1 + \langle q_1\|1, \mathbf{b}'[1]\|\mathbf{b}'[7]\rangle = a' \cdot \left(\langle q_1\|1, \mathbf{c}[1]\|\mathbf{c}[7]\rangle - p_1\right),$$
$$\pi_2 + \langle q_2\|1, \mathbf{b}'[3]\|\mathbf{b}'[8]\rangle = a' \cdot \left(\langle q_2\|1, \mathbf{c}[3]\|\mathbf{c}[8]\rangle - p_2\right),$$
$$\pi_3 + \langle q_1q_2\|q_1\|q_2\|1, \mathbf{b}'[5]\|\mathbf{b}'[9{:}11]\rangle = a' \cdot \left(\langle q_1q_2\|q_1\|q_2\|1, \mathbf{c}[5]\|\mathbf{c}[9{:}11]\rangle - p_1p_2\right).$$

P_j accepts if all the checks pass.

Note that, an honest proof is accepted because $a'\mathbf{a} = \mathbf{b} + \mathbf{b}'$ and

$$\begin{aligned}
\underset{\langle q_1\|1, \mathbf{b}[1]\|\mathbf{b}[7]\rangle}{\underset{\uparrow}{\pi_1}} + \langle q_1\|1, \mathbf{b}'[1]\|\mathbf{b}'[7]\rangle &= a' \cdot \langle q_1\|1, \mathbf{a}[1]\|\mathbf{a}[7]\rangle \\
&= a' \cdot \left(\langle q_1\|1, \mathbf{c}[1]\|\mathbf{c}[7]\rangle - \langle q_1\|1, v_1\|g_1\rangle\right) \\
&= a' \cdot \left(\langle q_1\|1, \mathbf{c}[1]\|\mathbf{c}[7]\rangle - p_1\right),
\end{aligned}$$

$$\overset{\langle q_1q_2\|q_1\|q_2\|1, \mathbf{b}[5]\|\mathbf{b}[9:11]\rangle}{\overset{\downarrow}{\pi_3}} + \langle q_1q_2\|q_1\|q_2\|1, \mathbf{b}'[5]\|\mathbf{b}'[9{:}11]\rangle = a' \cdot \langle q_1q_2\|q_1\|q_2\|1, \mathbf{a}[5]\|\mathbf{a}[9{:}11]\rangle$$
$$= a' \cdot \Big(\langle q_1q_2\|q_1\|q_2\|1, \mathbf{c}[5]\|\mathbf{c}[9{:}11]\rangle - \underbrace{\langle q_1q_2\|q_1\|q_2\|1, v_3\|v_1g_2\|g_1v_2\|g_1g_2\rangle}_{\substack{= \langle q_1q_2\|q_1\|q_2\|1, v_1v_2\|v_1g_2\|g_1v_2\|g_1g_2\rangle \\ = \langle q_1\|1, v_1\|g_1\rangle \cdot \langle q_2\|2, v_2\|g_2\rangle \\ = p_1p_2}}\Big).$$

The Protocol Π_{2MP^+}, Online Phase, ProveSame Command

Party P_i, on input $(\texttt{"ProveSame"}, \mathrm{var}, u)$, does the following.

Exceptional Case: Ignore the input if P_i has broadcast a message of form $(\texttt{"ProveSame"}, \mathrm{var}, \star, \star)$.

Prover's Message: To prove to every other party that it has committed to consistent values for var, P_i does the following:

- P_i prepares a linear function, represented by $q\|1$, by querying the random oracle with the set $S_{i,\mathsf{sent}}$ of commitments it has broadcast so far $q \leftarrow \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$. It broadcasts the output of the linear function on input $(u\|g_{i,\mathrm{var}})$, that is,
$$\text{broadcast } (\texttt{"ProveSame"},\ \mathrm{var},\ q,\ h = \langle q\|1, u\|g_{i,\mathrm{var}}\rangle) .$$
- For each $P_j \neq P_i$, party P_i enumerates over all previous commands, and sends
$$\text{to } P_j \left(\texttt{"ProveSame-0"},\ \mathrm{var},\ \perp\right)$$
if there is any $(\texttt{"Input"}, \mathsf{sid}_\ell = (\mathsf{2MP}, \{i,j\}, \mathsf{seqnum}), (\mathrm{var}, v_\ell))$ that $v_\ell \neq u$ or any $(\texttt{"ProveProd"}, \mathsf{sid}_\ell = (\mathsf{Pf}, i, j, \mathsf{seqnum}), (\mathrm{var}_{\ell,1}, v_{\ell,1}), (\mathrm{var}_{\ell,2}, v_{\ell,2}), (\mathrm{var}_{\ell,3}, v_{\ell,3}))$ that $\mathrm{var}_{\ell,t} = \mathrm{var}$ and $v_{\ell,t} \neq u$. Otherwise,
  - For each previous command $(\texttt{"Input"}, \mathsf{sid}_\ell = (\mathsf{2MP}, \{i,j\}, \mathsf{seqnum}), (\mathrm{var}, v_\ell))$ (satisfying $v_\ell = u$), party P_i retrieves the OLE correlated randomness $(\mathbf{a}, \mathbf{B})$ used in sid_ℓ from $S_{i,\mathsf{session}}$ and sends P_j
  $$\text{to } P_j \left(\texttt{"ProveSame-1"},\ \mathsf{sid}_\ell,\ \pi = \langle q\|1, \mathbf{B}[1{:}2, 3]\rangle\right)$$
  as a proof that the committed value in session sid_ℓ is consistent.
  - For each previous command $(\texttt{"ProveProd"}, \mathsf{sid}_\ell = (\mathsf{Pf}, i, j, \mathsf{seqnum}), (\mathrm{var}_{\ell,1}, v_{\ell,1}), (\mathrm{var}_{\ell,2}, v_{\ell,2}), (\mathrm{var}_{\ell,3}, v_{\ell,3}))$ that $\mathrm{var}_{\ell,t} = \mathrm{var}$ (and satisfies $v_{\ell,t} = u$), party P_i retrieves the OLE correlated randomness $(\mathbf{a}, \mathbf{b})$ used in sid_ℓ from $S_{i,\mathsf{prover}}$ and sends
  $$\text{to } P_j \left(\texttt{"ProveSame-2"},\ \mathsf{sid}_\ell,\ t,\ \pi = \langle q\|1, \mathbf{b}[2t-1:2t]\rangle\right) .$$

Verifier's Decision: Each $P_j \neq P_i$ decides to accept or reject as follows:

- Upon receiving the $\texttt{"ProveSame"}$ message, P_j collects the set S of messages P_i broadcast before the $\texttt{"ProveSame"}$ message, and verifies that the linear function $q\|1$ is evaluated correctly as $q = \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$.
- Upon receiving $(\texttt{"ProveSame-3"}, \mathrm{var}, \perp)$, P_j rejects.
- Upon receiving $(\texttt{"ProveSame-1"}, \mathsf{sid}_\ell, \pi)$, P_j retrieves the tensor-OLE correlated randomness $(\mathbf{a}', \mathbf{B}')$ used in sid_ℓ from $S_{j,\mathsf{session}}$, and the commitment $(\texttt{"Input"}, \mathsf{sid}_\ell, \mathrm{var}, \mathbf{c})$ it received from P_i before, and check if
$$\pi + \langle q\|1,\ \mathbf{B}'[3, 1{:}2]\rangle = \mathbf{a}'[3] \cdot (\langle q\|1,\ \mathbf{c}\rangle - h) .$$
Note that an honest proof is accepted because
$$\underset{\langle q\|1,\ \mathbf{B}[1:2,3]\rangle}{\pi} + \langle q\|1,\ \mathbf{B}'[3,1{:}2]\rangle = \langle q\|1,\ \mathbf{a}[1{:}2] \cdot \mathbf{a}'[3]\rangle = (\langle q\|1, \underset{(u\|g_{i,\mathrm{var}})+\mathbf{a}[1:2]}{\mathbf{c}}\rangle - \underset{\langle q\|1,\ u\|g_{i,\mathrm{var}}\rangle}{h}) \cdot \mathbf{a}'[3] \tag{8}$$
- Upon receiving $(\texttt{"ProveSame-2"}, \mathsf{sid}_\ell, t, \pi)$, P_j retrieves the vector-OLE correlated randomness $(a', \mathbf{b}')$ used in sid_ℓ from $S_{j,\mathsf{verifier}}$, and the commitment $\mathbf{c}$ it received from P_i before, and check if
$$\pi + \langle q\|1,\ \mathbf{b}'[2t-1:2t]\rangle = a' \cdot (\langle q\|1,\ \mathbf{c}[2t-1:2t]\rangle - h) .$$
Note that an honest proof is accepted because
$$\underset{\langle q\|1,\ \mathbf{b}[2t-1:2t]\rangle}{\pi} + \langle q\|1,\ \mathbf{b}'[2t-1:2t]\rangle = \langle q\|1,\ \mathbf{a}[2t-1:2t] \cdot a'\rangle = (\langle q\|1, \mathbf{c}\rangle - a' \cdot \underset{\langle q\|1,\ u\|g_{i,\mathrm{var}}\rangle}{h}) \tag{9}$$

P_j accept, if all checks pass.

The Protocol Π_{2MP^+}, Online Phase, 2MultPlus Commands

Party P_i and P_j on input $(\texttt{"2MultPlus"}, \mathsf{sid} = (\mathsf{2MP}, \{i,j\}, \mathsf{seqnum}), z_i)$ and $(\texttt{"2Multplus"}, \mathsf{sid}, z_j)$ respectively, do the following.

P_i's message: P_i retrieves information of session sid, including $(\mathsf{sid}, (\mathbf{a}_i, \mathbf{B}_i), \star, v_i)$ from $S_{i,\mathsf{session}}$ and the commitment $(\texttt{"Input"}, \mathsf{sid}, \star, \mathbf{c}_j)$ that P_j has broadcast before,

broadcasts $(\texttt{"2MultPlus"}, \mathsf{sid}, m_i = v_i\mathbf{c}_j[1] + \mathbf{B}_i[1,1] + z_i)$.

P_j's message: Similarly, P_j retrieves $(\mathsf{sid}, (\mathbf{a}_j, \mathbf{B}_j), \star, v_j)$ and $(\texttt{"Input"}, \mathsf{sid}, \star, \mathbf{c}_i)$,

broadcasts $(\texttt{"2MultPlus"}, \mathsf{sid}, m_j = v_j\mathbf{c}_i[1] + \mathbf{B}_j[1,1] + z_j)$.

Public output reconstruction: Every party P_k for $k \in [n]$, upon receiving the broadcast $\texttt{"2MultPlus"}$ messages, retrieves the commitment $(\texttt{"Input"}, \mathsf{sid}, \star, \mathbf{c}_i)$ from P_i and $(\texttt{"Input"}, \mathsf{sid}, \star, \mathbf{c}_j)$ from P_j, and outputs

$$(\texttt{"Output"}, \mathsf{sid}, m_i + m_j - \mathbf{c}_i[1] \cdot \mathbf{c}_j[1]) \ .$$

Note that the output is correct if P_i and P_j are honest as

$$\begin{aligned}
& m_i + m_j - \mathbf{c}_i[1] \cdot \mathbf{c}_j[1] \\
&= v_i \cdot \underset{\uparrow}{\mathbf{c}_j[1]} + \mathbf{B}_i[1,1] + z_i + v_j \cdot \underset{\uparrow}{\mathbf{c}_i[1]} + \mathbf{B}_j[1,1] + z_j - \underset{\uparrow}{\mathbf{c}_j[1]} \cdot \underset{\uparrow}{\mathbf{c}_i[1]} \\
&\quad\ v_j + \mathbf{a}_j[1] \qquad\qquad\qquad\quad v_i + \mathbf{a}_i[1] \qquad\qquad\qquad\quad v_j + \mathbf{a}_j[1] \quad v_i + \mathbf{a}_i[1] \\
&= v_i v_j + z_i + z_j + \mathbf{B}_i[1,1] + \mathbf{B}_j[1,1] - \mathbf{a}_i[1] \cdot \mathbf{a}_j[1] \\
&= v_i v_j + z_i + z_j
\end{aligned}$$

The functionality $\mathcal{F}_{\mathsf{2MP}^+}$ is securely implemented by the above protocol. In the rest of the section, we explicitly present the simulator and prove the simulation correctness.

Data Structure: For each corrupted $P_i \in C$, the simulator prepares the following data structure, which is of the similar structure as the honest parties' private storage.

- Set $S_{i,\mathsf{session}}$ will contain information for every computation session sid P_i participates in, including the OLE correlation $(\mathbf{a}, \mathbf{B})$, the input variable var and plain text $\mathbf{w} \in \mathbb{F}^2$, that is, $S_{i,\mathsf{session}} = \{(\mathsf{sid}, (\mathbf{a}, \mathbf{B}), \mathrm{var}, \mathbf{w})\}_{\mathsf{sid}}$. Initially, only $\mathbf{a}, \mathbf{B}$ are set, other values are uninitialized.
- Set $S_{i,\mathsf{prover}}$ will contain information for every proof session sid P_i participates in as the prover, including the OLE correlation $(\mathbf{a}, \mathbf{b})$, the input variables $\mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3$ and the plain text $\mathbf{w} \in \mathbb{F}^{11}$, that is, $S_{i,\mathsf{prover}} = \{(\mathsf{sid}, (\mathbf{a}, \mathbf{b}), \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{w})\}_{\mathsf{sid}}$. Initially, only $\mathbf{a}, \mathbf{b}$ are set, other values are uninitialized.
- Set $S_{i,\mathsf{verifier}}$ will contain information for every proof session sid P_i participates in as the verifier, which is the OLE correlation $(a, \mathbf{b})$, that is, $S_{i,\mathsf{verifier}} = \{(\mathsf{sid}, (a, \mathbf{b}))\}_{\mathsf{sid}}$.

Note that, the simulator knows any corrupted party's portion of correlated randomness, as they can be extracted from the adversary's view in the offline phase.

The simulator also keeps a record $S_{i,\mathsf{sent}}$ of all the broadcast messages from P_i. For each corrupted $P_i \in C$, all broadcast messages are recorded. For each uncorrupted $P_i \notin C$, all *simulated* broadcast messages are recorded.

Input command. Upon the input command, the party broadcasts the one-time pad of a party-generated vector. The one-time pad key comes from tensor-OLE correlation, which is known to the simulator. Thus the simulation is simple: To simulate an honest party, its message is a random string. To extract from a corrupt party, the simulator decrypts the one-time pad, stores the plain text for later use, and extracts the effective ideal functionality call since it is written in the plain text. Formally, the simulator works as follows:

Extract from corrupted party's messages When a corrupted party $P_i \in C$ broadcasts a message $(\texttt{"Input"}, \mathsf{sid}, \mathrm{var}, \mathbf{c})$, the session id $\mathsf{sid} = (\{i, j\}, \mathsf{seqnum})$ and the variable name var are revealed in plain.

The simulator retrieves OLE correlation $\mathbf{a} \in \mathbb{F}^3, \mathbf{B} \in \mathbb{F}^{3\times 3}$ from $S_{i,\mathsf{session}}$, recovers the underlying message $\mathbf{w} = \mathbf{c} - \mathbf{a}[1{:}2]$, updates $(\mathsf{sid}, P_i, \mathrm{var}, \mathbf{w})$ in $S_{i,\mathsf{session}}$ and extracts command

$$\Big(\texttt{"Input"},\ \mathsf{sid},\ (\mathrm{var},\ \mathbf{w}[1])\Big).$$

Simulate honest party's messages When the functionality sends message $(\mathsf{sid}, P, \mathrm{var}, \texttt{"Received"})$ for $P \notin C$. The simulator samples random $\mathbf{c} \in \mathbb{F}^2$ and simulates the corresponding broadcast message from P as

$$\Big(\texttt{"Input"},\ \mathsf{sid},\ \mathrm{var},\ \mathbf{c}\Big).$$

The simulation works since $\mathbf{c}$ in the real world is an one-time pad.

ProveProd command. The prover P_i want to convince the verifier P_j that the input variables $(\mathrm{var}_1, v_1), (\mathrm{var}_2, v_2), (\mathrm{var}_3, v_3)$ satisfies that $v_1 v_2 = v_3$. Such proof is enabled by the ProveProd command, which consists of two components.

- commit-and-prove-linear: The prover P_i broadcasts an OTP, which serves as the commitment of a vector $\mathbf{w} = (v_1, g_{i,\mathrm{var}_1}, v_2, g_{i,\mathrm{var}_2}, v_3, g_{i,\mathrm{var}_3}, \dots)$. As an OTP, this message is easy to simulate.
- Linear PCP: The OTP allows P_i to securely reveal any linear function on $\mathbf{w}$ to the verifier P_j. Therefore, linear PCP allows the P_i to prove any arithmetic condition of $\mathbf{w}$. The proof requires $\mathbf{w}$ to be appended by some P_i-generated elements.

Extract from corrupted party's messages When corrupted $P_i \in C$ broadcasts $(\texttt{"ProveProd"}, \mathsf{sid}, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{c}, q_1, q_2, p_1, p_2, \pi_1, \pi_2, \pi_3)$, the simulator retrieves $(\mathbf{a}, \mathbf{b})$ from set $S_{i,\mathsf{prover}}$, computes the underlying plain text $\mathbf{w} := \mathbf{c} - \mathbf{a}$, replaces the entry with $(\mathsf{sid}, (\mathbf{a}, \mathbf{b}), \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{w})$ in the set, and extracts command

$$(\texttt{"ProveProd"}, \mathsf{sid}, (\mathrm{var}_1, \mathbf{w}[1]), (\mathrm{var}_2, \mathbf{w}[3]), (\mathrm{var}_3, \mathbf{w}[5])).$$

To decide the value of 1_{rej}, the simulator tests the following.

1. Check if (q_1, q_2) are evaluated correctly as $(q_1, q_2) \leftarrow \mathsf{RO}(\mathsf{sid}, \mathbf{c})$.
2. Check if $p_1 = \langle q_1 \| 1,\ \underset{\text{should be } v_1\|g_1}{\mathbf{w}[1]\|\mathbf{w}[7]}\rangle$, $p_2 = \langle q_2 \| 1,\ \underset{\text{should be } v_2\|g_2}{\mathbf{w}[3]\|\mathbf{w}[8]}\rangle$, and $p_1 p_2 = \langle q_1 q_2 \| q_1 \| q_2 \| 1,\ \underset{\text{should be } v_3\|v_1 g_2\|g_1 v_2\|g_1 g_2}{\mathbf{w}[5]\|\mathbf{w}[9{:}11]}\rangle$.
3. Check if $\pi_1 = \langle q_1 \| 1,\ \mathbf{b}[1]\|\mathbf{b}[7]\rangle$, $\pi_2 = \langle q_2 \| 1,\ \mathbf{b}[3]\|\mathbf{b}[8]\rangle$, and $\pi_3 = \langle q_1 q_2 \| q_1 \| q_2 \| 1,\ \mathbf{b}[5]\|\mathbf{b}[9{:}11]\rangle$.
4. Check if $\mathbf{w}[1] \cdot \mathbf{w}[3] = \mathbf{w}[5]$. Note that $\mathbf{w}[1], \mathbf{w}[3], \mathbf{w}[5]$ are the extracted values of variables $\mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3$ respectively.

If one of the first three checks fails, the simulator feed $1_{\mathsf{rej}} = 1$ when the functionality asks for additional input. If all checks pass, the simulator feed $1_{\mathsf{rej}} = 0$ when the functionality asks for additional input. If the first three checks all pass, but the last check fails, the simulator *gives up*.

We verify the simulation case by case.

- *One of the first three tests fails.* We prove that the honest verifier will reject in such case, thus setting $1_{\mathsf{rej}} = 1$ is the right simulation. If the 1st check fails, the verifier can directly detect it. If the 2nd check fails, the prover is trying to prove a false linear condition on the committed vector $\mathbf{w}$, which will be caught by the verifier with overwhelming probability. If the 2nd check passes but the 3rd check fails, the prover is trying to prove a correctly linear condition using a wrong proof, which will be rejected with overwhelming probability.
- *All checks pass.* By similar argument, the verifier will accept in such case. Since $\mathbf{w}[1] \cdot \mathbf{w}[3] = \mathbf{w}[5]$, and 1_{rej} is set to 0, the verifier will receive acc in the ideal world as well. Note that $\mathbf{w}[1], \mathbf{w}[3], \mathbf{w}[5]$ are the extracted values of variables $\mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3$ respectively.
- *The first three checks all pass, but the last check fails.* By similar argument, the verifier will accept in such case. But the verifier will receives rej in the ideal world despite the value of 1_{rej}, because the last check fails. We have to prove that such case happens with only negligible probability.

In particular, we are going to bound the probability that *the first two checks pass but the last check fails*. The second check passes requires that

$$\langle q_1 \| 1,\ \underset{\substack{\uparrow\\ \text{extracted value of } \mathrm{var}_1}}{\mathbf{w}[1]} \| \mathbf{w}[7] \rangle \cdot \langle q_2 \| 1,\ \underset{\substack{\uparrow\\ \text{extracted value of } \mathrm{var}_2}}{\mathbf{w}[3]} \| \mathbf{w}[8] \rangle = \langle q_1 q_2 \| q_1 \| q_2 \| 1,\ \underset{\substack{\uparrow\\ \text{extracted value of } \mathrm{var}_3}}{\mathbf{w}[5]} \| \mathbf{w}[9{:}11] \rangle$$

which can be rewritten as a quadratic polynomial on (q_1, q_2) being 0

$$(\mathbf{w}[5] - \mathbf{w}[1]\mathbf{w}[3]) \cdot q_1 q_2 + (\mathbf{w}[9] - \mathbf{w}[1]\mathbf{w}[8]) \cdot q_1 + (\mathbf{w}[10] - \mathbf{w}[7]\mathbf{w}[3]) \cdot q_2 + (\mathbf{w}[11] - \mathbf{w}[7]\mathbf{w}[8]) = 0. \quad (10)$$

If $\mathbf{w}[5] \neq \mathbf{w}[1]\mathbf{w}[3]$, the quadratic polynomial is non-zero, thus (10) is not satisfied with overwhelming probability as long as q_1, q_2 are randomly sampled by the random oracle.

The above analysis works if the adversary does not queries the random oracle before P_i broadcasts the message. In order to bound the probability the simulator gives up, we have to show that querying the random oracle beforehand will not benefit the adversary dramatically. Thus we need a stronger statement here:

> The adversary cannot find tuple $(\mathsf{sid}, \mathbf{w}, \mathbf{a})$ such that the other party in session sid is not corrupted, $(\mathbf{a}, \mathbf{b})$ is P_i's portion of OLE correlated randomness under session sid, and Equation (10) holds for $(q_1, q_2) = \mathsf{RO}(\mathsf{sid}, \mathbf{c} = \mathbf{w} + \mathbf{a})$.

The proof of the statement goes as follows. Since the adversary has no control on $\mathbf{a}$ if the other party in session sid is honest, we can assume w.l.o.g. that $\mathbf{a}$ has been generated by the OLE correlation generation protocol $\mathcal{F}_{\mathsf{OLEcor}}$ before the adversary queries $\mathsf{RO}(\mathsf{sid}, \star)$. Whenever the adversary queries $\mathsf{RO}(\mathsf{sid}, \mathbf{c})$, the corresponding plain text $\mathbf{w}$ is determined by $\mathbf{w} = \mathbf{c} - \mathbf{a}$. Thus due to the randomness of $(q_1, q_2) \leftarrow \mathsf{RO}(\mathsf{sid}, \mathbf{c})$, Equation (10) holds with probability $O(\frac{1}{|\mathbb{F}|})$. Thus the adversary finds such $(\mathsf{sid}, \mathbf{w}, \mathbf{a})$ with negligible probability, as long as it only queries the random oracle polynomial number of times.

Simulate honest party's messages When the functionality sends message $(\mathsf{sid}, P, \mathrm{var}, \mathsf{rej})$ to $P_j \in C$ on behalf of honest P, the simulator simulates P's messages as

$$(\texttt{"ProveProd"}, \mathsf{sid}, \bot).$$

The simulation works because an honest party will never try to prove a false statement.

When the functionality sends message $(\mathsf{sid}, P, \mathrm{var}, \mathsf{acc})$ to $P_j \in C$ on behalf of honest P, the simulator samples $\mathbf{c} \leftarrow \mathbb{F}^{11}$, computes $(q_1, q_2) \leftarrow \mathsf{RO}(\mathsf{sid}, \mathbf{c})$, samples $p_1, p_2 \leftarrow \mathbb{F}$. Then the simulator retrieves $a', \mathbf{b}'$ from $S_{j,\mathsf{verifier}}$, and simulates P's message as

$$\Big(\texttt{"ProveProd"},\ \mathsf{sid},\ \mathrm{var}_1,\ \mathrm{var}_2,\ \mathrm{var}_3,\ \mathbf{c},\ q_1,\ q_2,\ p_1,\ p_2,\ \pi_1,\ \pi_2,\ \pi_3\Big).$$

where π_1, π_2, π_3 are computed from

$$\begin{aligned}
\pi_1 + \langle q_1 \| 1, \mathbf{b}'[1] \| \mathbf{b}'[7] \rangle &= a'\big(\langle q_1 \| 1, \mathbf{c}[1] \| \mathbf{c}[7] \rangle - p_1\big), \\
\pi_2 + \langle q_2 \| 1, \mathbf{b}'[3] \| \mathbf{b}'[8] \rangle &= a'\big(\langle q_2 \| 1, \mathbf{c}[3] \| \mathbf{c}[8] \rangle - p_2\big), \\
\pi_3 + \langle q_1 q_2 \| q_1 \| q_2 \| 1, \mathbf{b}'[5] \| \mathbf{b}'[9{:}11] \rangle &= a'\big(\langle q_1 q_2 \| q_1 \| q_2 \| 1, \mathbf{c}[5] \| \mathbf{c}[9{:}11] \rangle - p_1 p_2\big).
\end{aligned}$$

The simulation works, because in the real world, $\mathbf{c}, q_1, q_2$ are one-time padded by $\mathbf{a}, g_1, g_2$ respectively, and π_1, π_2, π_3 satisfy the above equations.

ProveSame command. Say party P_i has (var, v) as its input variable. Party P_i samples a random *guard* $g_{\mathrm{var}} \in \mathbb{F}$, such that in every session sid in which var is involved, P_i will commit to a vector containing v, g_{var} using the commit-and-prove-linear mechanism.

Upon the ProveSame command, P_i queries the random oracle $q \leftarrow \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$ to get a random challenge $q \in \mathbb{F}$. It uses the commit-and-prove-linear mechanism to reveal $qv + g_{\mathrm{var}}$ on every session. Assume that corrupt P_i committed to inconsistent v', g' in some session, then $qv' + g' \neq qv + g_{\mathrm{var}}$ with overwhelming probability due to the randomness of q, thus the corruption will be detected.

For example, if var is involved in a proof session $\mathsf{sid} = (i, j, \mathsf{seqnum})$. The two active parties P_i, P_j jointly hold vector-OLE correlated randomness. Party P_i receives $\mathbf{a}_i, \mathbf{b}_i$, party P_j receives $a_j, \mathbf{b}_j$, such that $\mathbf{a}_i \cdot a_j = \mathbf{b}_i + \mathbf{b}_j$. Upon the ProveProd command, P_i sends the OTP $\mathbf{c} = (\dots, v, g_{\mathrm{var}}, \dots) + \mathbf{a}$ as the commitment of a vector containing v, g_{var}.

Similarly, if var is involved in a computation session $\mathsf{sid} = (\{i, j\}, \mathsf{seqnum})$. The two active parties P_i, P_j jointly hold tensor-OLE correlated randomness. Party P_i receives $\mathbf{a}_i, \mathbf{B}_i$, party P_j receives $\mathbf{a}_j, \mathbf{B}_j$, such that $\mathbf{a}_i \mathbf{a}_j^\intercal = \mathbf{B}_i + \mathbf{B}_j^\intercal$. The ProveSame command critically relies on a vector-OLE correlated randomness embed inside the tensor-OLE correlated randomness. In particular, the correlated randomness consists of $\mathbf{a}_i[1{:}2], \mathbf{B}_i[1{:}2, 3]$ hold by P_i and $\mathbf{a}_j[3], \mathbf{B}_j[3, 1{:}2]$ hold by P_j. They jointly form a vector-OLE correlation as

$$\mathbf{a}_i[1{:}2] \cdot \mathbf{a}_j[3] = \mathbf{B}_i[1{:}2, 3] + \mathbf{B}_j[3, 1{:}2].$$

Upon the Input command, P_i sends the OTP $\mathbf{c} = (v, g_{\mathrm{var}}) + \mathbf{a}_i[1{:}2]$ as the commitment of (v, g_{var}).

In either case, the commitment allows P_i to later reveal a linear function on (v, g_{var}) to P_j.

Extract from corrupted party's messages When $(\texttt{"ProveSame"}, \mathrm{var}, q, h)$ is broadcast by a corrupted party $P_i \in C$. W.l.o.g. we assume q is evaluated correctly as $q \leftarrow \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$. The simulator calls command

$$(\texttt{"ProveSame"},\ \mathrm{var},\ v),$$

and feed indicator vector $\mathbf{1}_{\mathsf{rej}} \in \{0, 1\}^n$ to the functionality as additional input. The simulator extracts v and $\mathbf{1}_{\mathsf{rej}}$ as follows.

- Initialize $\mathbf{1}_{\mathsf{rej}}$ as all zeros. Start with v being uninitialized.
- Upon P_i sends $(\texttt{"ProveSame-0"}, \mathrm{var}, \perp)$ to P_j, set $\mathbf{1}_{\mathsf{rej}}[j] = 1$.
- For every $(\mathsf{sid}, (\mathbf{a}, \mathbf{B}), \mathrm{var}, \mathbf{w})$ in set $S_{i,\mathsf{session}}$ that has the same variable var, say P_j is the other party in the session, party P_i also sends $(\texttt{"ProveSame-1"}, \mathsf{sid}, \pi)$ to P_j. (Otherwise P_j will reject, and the simulator can correctly simulate it by setting $\mathbf{1}_{\mathsf{rej}}[j] = 1$.)
  The simulator checks if $q \cdot \mathbf{w}[1] + \mathbf{w}[2] = h$ and $\pi = \langle q \| 1,\ \mathbf{B}[1{:}2, 3] \rangle$.
  - If either check fails, P_j will reject the proof, thus the simulator should set $\mathbf{1}_{\mathsf{rej}}[j] = 1$. *Note that the simulator can predict whether the message will cause rejection. P_j will accept if both checks pass, and will reject with overwhelming probability if either check fails.*
  - If both checks pass, and v is not initialized yet, the simulator sets $v \leftarrow \mathbf{w}[1]$. *Note that the simulator had extracted* $\mathbf{w}[1]$ *as value of* var *from P_i's previous broadcast message in computation session* sid.
  - If both checks pass, and $v = \mathbf{w}[1]$, the simulator does nothing.
  - If both checks pass, and $v \neq \mathbf{w}[1]$, the simulator *gives up*.
- For every $(\mathsf{sid}, (\mathbf{a}, \mathbf{b}), \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{w})$ in $S_{i,\mathsf{prover}}$ and $t \in \{1, 2, 3\}$ such that $\mathrm{var}_t = \mathrm{var}$, say P_j is the other party in the session, party P_i also sends $(\texttt{"ProveSame-2"}, \mathsf{sid}, t, \pi)$ to P_j. (Otherwise P_j will reject, and the simulator can correctly simulate it by setting $\mathbf{1}_{\mathsf{rej}}[j] = 1$.)
  Check if $q \cdot \mathbf{w}[2t-1] + \mathbf{w}[2t] = h$ and $\pi = \langle q \| 1,\ \mathbf{b}[2t-1 : 2t] \rangle$.
  - If either check fails, P_j will reject the proof, thus the simulator should set $\mathbf{1}_{\mathsf{rej}}[j] = 1$. *Note that the simulator can predict whether the message will cause rejection. P_j will accept if both checks pass, and will reject with overwhelming probability if either check fails.*
  - If both checks pass, and v is not initialized yet, the simulator sets $v \leftarrow \mathbf{w}[2t-1]$. *Note that the simulator had extracted* $\mathbf{w}[2t-1]$ *as value of* var *from P_i's previous broadcast message in proof session* sid.
  - If both checks pass, and $v = \mathbf{w}[2t-1]$, the simulator does nothing.
  - If both checks pass, and $v \neq \mathbf{w}[2t-1]$, the simulator *gives up*.
- If v is still not initialized, set $v \leftarrow 0$.

In short, the simulator works as follows. For every proof sent by the corrupted P_i – either $(\texttt{"ProveSame-1"}, \mathsf{sid}, \pi)$ or $(\texttt{"ProveSame-2"}, \mathsf{sid}, t, \pi)$ – the simulator can predict whether the proof will be accepted or not. If not, correct simulation is as easy as setting a bit in the indicator vector $\mathbf{1}_{\mathsf{rej}}$ to be one. If the proof will be accepted, then the simulator recalls the extracted value of var in the corresponding session, stores it in v, and extracts command

$$(\texttt{"ProveSame"},\ \mathrm{var},\ v)$$

to claim that v is the right value of var.

The only problem is that, there may be another proof from P_i that will also be accepted, but the extracted value of var in the corresponding session is different from v. In such case, the simulator gives up.

As long as the simulator does not give up, P_i must have used the same value for all appearance of variable var in the interaction with all accepting parties.

To complete the argument, we need to show that the simulator gives up with negligible probability. Intuitively, if P_i commits to $v_1 \| g_1$ in one session and commits to $v_2 \| g_2$ in another, then $\langle q \| 1, v_1 \| g_1 \rangle = \langle q \| 1, v_2 \| g_2 \rangle$ with overwhelming probability if q is randomly sampled. But if q is sampled by the random oracle, we need to additionally argue that P_i cannot choose v_1, g_1, v_2, g_2 depending on the output of the random oracle. We formalize this Fiat-Shamir arguing by showing a stronger statement:

> The adversary cannot find a tuple $(P_i, \mathrm{var}, S_{i,\mathsf{sent}}, q)$ consisting of a party $P_i \in C$, a set of messages $S_{i,\mathsf{sent}}$, a variable var, and $q = \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$, such that by defined a set $\{(v_\ell \| g_\ell)\}_\ell$ containing
>
> - $(v_\ell \| g_\ell) = \mathbf{c} - \mathbf{a}$ for each exists $(\texttt{"Input"}, \mathsf{sid}, \mathrm{var}, \mathbf{c}) \in S_{i,\mathsf{sent}}$, such that the other party in session sid is not corrupted, and $(\mathbf{a}, \mathbf{B})$ is P_i's portion of tensor-OLE correlation in the session.
> - $(v_\ell \| g_\ell) = \mathbf{c}[2t-1:2t] - \mathbf{a}[2t-1:2t]$ for each $t \in \{1,2,3\}$ and $(\texttt{"ProveSum"}, \mathsf{sid}, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{c}, \dots) \in S_{i,\mathsf{sent}}$ such that the other party in session sid is not corrupted, $\mathrm{var}_t = \mathrm{var}$ and $(\mathbf{a}, \mathbf{b})$ is P_i's portion of vector-OLE correlation in the session.
>
> The linear mapping $q \| 1$ has a collision on the set $\{(v_\ell \| g_\ell)\}_\ell$. That is, $\langle q \| 1,\ v_\ell \| g_\ell \rangle = \langle q \| 1,\ v_{\ell'} \| g_{\ell'} \rangle$ for distinct $(v_\ell \| g_\ell), (v_{\ell'} \| g_{\ell'})$ in the set.

The proof of the above statement goes as follows. Since the adversary has no control on $\mathbf{a}$ if the other party in session sid is honest, we can assume w.l.o.g. that $\mathbf{a}$ has been generated by $\mathcal{F}_{\mathsf{OLEcor}}$ for every session occurs in $S_{i,\mathsf{sent}}$, before the adversary queries $\mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$.

Whenever the adversary queries $\mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$, the corresponding set $\{(v_\ell \| g_\ell)\}_\ell$ is determined before the query. Thus due to the randomness of q, there is no collision with overwhelming probability by birthday bound, as long as $S_{i,\mathsf{sent}}$ contains polynomially many messages. Thus the adversary finds such tuple $(P_i, \mathrm{var}, S_{i,\mathsf{sent}}, q)$ with negligible probability, as long as it only queries the random oracle polynomially many times.

REMARK: A TIGHTER SECURITY ANALYSIS. Simulation for ProveSame command will be the security level bottleneck, thus we provide a careful and tighter analysis here.

Say the corrupted party P_i, has committed to the values var in sessions with P_j for m_j times. That is, P_i has committed to $v_{j,\ell} \| g_{j,\ell}$ in a session with P_j for each $\ell \le m_j$.

In the current analysis, the simulator will give up if there exists $(j, \ell) \neq (j', \ell')$ such that $\langle q \| 1, v_{j,\ell} \| g_{j,\ell} \rangle = \langle q \| 1, v_{j',\ell'} \| g_{j',\ell'} \rangle$ and $v_{j,\ell} \neq v_{j',\ell'}$. Therefore, the error probability is $O(\frac{(m_1+\cdots+m_n)^2}{|\mathbb{F}|})$, if q randomly chosen. If the adversary repeatedly asks RO to provide t_{RO} different q's for the variable name var, the error probability will grow proportionally to $O(\frac{(m_1+\cdots+m_n)^2}{|\mathbb{F}|} \cdot t_{\mathsf{RO}})$.[8] We can improve the simulation and remove the dependence on $m_1 + \cdots + m_n$ from the error probability.

The current simulator gives up too easily. For example, say the corrupt party chooses distinct $v_{j,1}, v_{j,2}, v_{j,3}$. Then on the rare event

$$\langle q \| 1, v_{j,1} \| g_{j,1} \rangle = \langle q \| 1, v_{j,2} \| g_{j,2} \rangle \neq \langle q \| 1, v_{j,3} \| g_{j,3} \rangle,$$

the simulator will give up because the corrupted P_i manage to mislead P_j into believing $v_{j,1} = v_{j,2}$. But the simulator does not need to give up on this event. P_j will correctly reject the proof anyway since it detects $v_{j,2} \neq v_{j,3}$.

A better and still sufficient set of criteria for giving up, is the following:

[8] If we assume w.l.o.g. that the adversary uses one of the RO outcomes it queried. Otherwise t_{RO} should be replace by $t_{\mathsf{RO}} + 1$.

- *Criterion 1:* For each j, if $v_{j,1}, \ldots, v_{j,\ell_j}$ are not all equal and

$$\langle q \| 1, v_{j,1} \| g_{j,1} \rangle = \langle q \| 1, v_{j,2} \| g_{j,2} \rangle = \cdots = \langle q \| 1, v_{j,\ell} \| g_{j,\ell} \rangle ,$$

  then the simulator should give up.

- *Criterion 2:* If there exists $j \neq j'$ such that $v_{j,1} \neq v_{j',1}$

$$\langle q \| 1, v_{j,1} \| g_{j,1} \rangle = \langle q \| 1, v_{j',1} \| g_{j',1} \rangle ,$$

  then the simulator should give up.

When both criteria are not met, the simulator can safely extract $v = v_{j,1}$ as the value of var, for any j s.t. $\langle q \| 1, v_{j,1} \| g_{j,1} \rangle = h$.

For a randomly sampled q, for each j, criterion 1 is met with probability at most $\frac{1}{|\mathbb{F}|}$, since there exists at most one value of q satisfying the equation. Criterion 1 is met with probability $O(\frac{n^2}{|\mathbb{F}|})$, by birthday bound. Overall, each adversary's query to $\mathsf{RO}(P_i, \mathrm{var}, \star)$ may increase the error probability by $O(\frac{n^2}{|\mathbb{F}|})$. The total error probability, caused by ProdSame command on var, is $O(\frac{n^2}{|\mathbb{F}|} \cdot t_{\mathsf{RO}})$.

Simulate honest party's messages When the functionality sends $(P_i, \mathrm{var}, \star)$ to all corrupted parties on behalf of an uncorrupted P_i, the simulator computes $q \leftarrow \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$, samples a random $h \in \mathbb{F}$, and simulates P_i's broadcast message as

$$\Big(\texttt{"ProveSame"}, \mathrm{var}, q, h\Big) .$$

In the real world, h is also uniform because it is one-time padded by $g_{i,\mathrm{var}}$.

If the functionality sends $(P_i, \mathrm{var}, \mathsf{rej})$ to a corrupted P_j, simulate P_i's message to P_j as

$$(\texttt{"ProveSame"}, \mathrm{var}, \bot).$$

Note that, this correctly simulates the honest party's message, because an honest party never try to forge a proof.

Otherwise, if the functionality sends $(P_i, \mathrm{var}, \mathsf{acc})$ to a corrupted P_j, the simulator queries $q \leftarrow \mathsf{RO}(P_i, \mathrm{var}, S_{i,\mathsf{sent}})$.

- For every $(\mathsf{sid}, P_i, \mathrm{var}, \mathbf{c}) \in S_{i,\mathsf{sent}}$ such that P_j is the other party in session sid, the simulator retrieves P_j's portion of tensor-OLE correlation $(\mathbf{a}', \mathbf{B}')$ used in session sid from $S_{j,\mathsf{session}}$, and simulates P_i's message as

$$(\texttt{"ProveSame-1"}, \mathsf{sid}, \pi)$$

  where π is uniquely determined by (8).

- For every $(\mathsf{sid}, P_i, \mathrm{var}_1, \mathrm{var}_2, \mathrm{var}_3, \mathbf{c}, \ldots) \in S_{i,\mathsf{sent}}$ and $t \in \{1, 2, 3\}$ such that P_j is the other party in session sid and $\mathrm{var}_t = \mathrm{var}$, the simulator retrieves P_j's portion of vector-OLE correlation $(a', \mathbf{b}')$ used in session sid from $S_{j,\mathsf{session}}$, and simulates P_i's message as

$$(\texttt{"ProveSame-2"}, \mathsf{sid}, t, \pi)$$

  where π is uniquely determined by (9).

Note that, this correctly simulates the honest party's message. Because once an honest party sends a proof, the proof must be correct.

2MultPlus command. Because we are considering the security with output substitution, and because the output of 2MultPlus is publicly reconstructable, it is sufficient to show that the honest parties' messages can be simulated. No matter what the corrupted parties output, the simulator can extract the output due to public output reconstruction, and simulate it using the output substitution ability.

Extract from void As we are concerning the security with output substitution, the simulator automatically extracts command

$$(\texttt{"2MultPlus"},\ \mathsf{sid},\ z_i = 0)$$

on behalf of corrupted P_i, for every computation session sid that P_i participates in.

Simulate honest party's messages When the functionality sends $(\texttt{"Output"}, \mathsf{sid}, y)$ to the adversary. Say the session id $\mathsf{sid} = (\{i, j\}, \mathsf{seqnum})$. The simulator retrieves P_i's broadcast message $(\texttt{"Input"}, \star, \star, \mathbf{c}_i)$ from $S_{i,\mathsf{session}}$, and P_j's broadcast message $(\texttt{"Input"}, \star, \star, \mathbf{c}_j)$ from $S_{j,\mathsf{session}}$. The simulation depends on whether P_i and/or P_j is corrupted.

- If both P_i, P_j are uncorrupted, samples random $m_i, m_j \in \mathbb{F}$ such that $m_i + m_j = \mathbf{c}_i[1] \cdot \mathbf{c}_j[1] + y$, simulates their messages as

$$(\texttt{"2MultPlus"}, \mathsf{sid}, m_i), \quad (\texttt{"2MultPlus"}, \mathsf{sid}, m_j).$$

- If one of P_i, P_j is uncorrupted. W.l.o.g., assume P_i is uncorrupted and P_j is corrupted. The simulator retrieves $(\mathsf{sid}, (\mathbf{a}, \mathbf{B}), \star, \mathbf{w})$ from $S_{j,\mathsf{session}}$, and simulates P_i's message as

$$(\texttt{"2MultPlus"}, \mathsf{sid}, m_i),$$

  where m_i is determined by

$$m_i = \mathbf{c}_i[1] \cdot \mathbf{c}_j[1] + y - \underbrace{(\mathbf{w}_j[1]\mathbf{c}_j[1] + \mathbf{B}_j[1,1])}_{\text{honest } P_j\text{'s message on } z_j = 0}.$$

- If both P_i, P_j are corrupted, there is no need to simulate honest party's message.

Output Substitution No matter what messages are sent by the corrupted parties, the simulator can easily compute substitution output, due to public output reconstruction.

- If one of P_i, P_j is corrupted, w.l.o.g., assume P_j is corrupted. P_j is allowed to adaptively broadcast

$$(\texttt{"2MultPlus"}, \mathsf{sid}, m_j)$$

  after receiving all honest parties' messages. By doing so, the adversary rewrite the output of this session as $y' = m_i + m_j - \mathbf{c}_i[1] \cdot \mathbf{c}_j[1]$.
- If both P_i, P_j are corrupted, P_i, P_j are allowed to adaptively broadcast

$$(\texttt{"2MultPlus"}, \mathsf{sid}, m_i), \quad (\texttt{"2MultPlus"}, \mathsf{sid}, m_j).$$

  after receiving all honest parties' messages. By doing so, the adversary rewrite the output of this session as $y' = m_i + m_j - \mathbf{c}_i[1] \cdot \mathbf{c}_j[1]$.

Statistical Security Loss. The total security loss is the sum of: (a) the probability that the simulator gives up; (b) the soundness error of the commit-and-prove-linear mechanism.

Each time commit-and-prove-linear mechanism is used to prove a false statement, there is an $\frac{1}{|\mathbb{F}|}$ soundness error. Notice that in our protocol, every instance of OLE correlated randomness is used by $O(1)$ commitment, and only $O(1)$ statements will be proved regarding each commitment. Thus the total soundness error is no more than $O(\frac{\text{the number of OLE correlated randomness instances}}{|\mathbb{F}|})$.

As we have analyzed, each adversary's query to $\mathsf{RO}(P_i, \mathrm{var}, \star)$ may increase the probability of giving up by at most $O(\frac{n^2}{|\mathbb{F}|})$. Similarly, each adversary's query to $\mathsf{RO}(\mathsf{sid}, \star)$ may increase the giving up probability by at most $O(\frac{1}{|\mathbb{F}|})$. In total, the probability that the simulator gives up is no more than $O(\frac{n^2}{|\mathbb{F}|} \cdot Q_{\mathsf{RO}})$, where Q_{RO} is the number of random oracle queries that the adversary makes.

6 MPC for Degree-3 Function

Lemma 6.1. *For any degree-3 n-party function $f : \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_n} \to \mathbb{F}^{\ell}$, there is a 2-round statistically maliciously secure MPC protocol that implements $\mathcal{F}_f^{\mathsf{os}}$, using tensor-OLE correlated randomness. The security level is $\frac{\mathrm{poly}(\lambda)}{|\mathbb{F}|}$ if $\ell, \ell_1, \ldots, \ell_n, n$ are at most polynomial in λ.*

Proof. The lemma follows from the composition of two results:

- For any degree-3 function f, there is an effective-degree-2 semi-malicious MPRE that has arithmetic preprocessing and uses tensor-OLE correlated randomness (Lemma 4.1).
- For any effective-degree-2 function g, $\mathcal{F}_g^{\mathsf{os}}$ is implemented by a 2-round maliciously secure MPC protocol using tensor-OLE correlated randomness (Lemma 5.1),

The composition comes from the definition of the semi-malicious secure MPRE (Def. 5). □

Efficiency Analysis. Let $\mathsf{mc}(f)$ denote the total number of monomials in f.

As we constructed and analyzed in Sec. 4.3, every degree-3 function f has a semi-malicious MPRE $(\hat{f}, h_1, \ldots, h_n)$ whose complexity is $O(\mathsf{mc}(f))$. In more detail, its encoding function satisfies $\mathsf{mc}(\hat{f}) = O(\mathsf{mc}(f))$, its preprocessing functions satisfy $\mathsf{cs}(h_1) + \cdots + \mathsf{cs}(h_n) = O(\mathsf{mc}(f))$, and it consumes $O(\mathsf{mc}(f))$ instances of 4×4 OLE correlated randomness,

As we analyzed in Sec. 5.1, there is a MPC protocol securely implementing $\mathcal{F}_{\hat{f} \circ h}^{\mathsf{os}}$ that consumes $O(\mathsf{mc}(f))$ instances of constant-size OLE correlated randomness, sends $O(\mathsf{mc}(f))$ field elements in P2P channels, and broadcasts $O(\mathsf{mc}(f))$ field elements.

Composing them together, there is a MPC protocol securely implementing $\mathcal{F}_f^{\mathsf{os}}$ against malicious corruptions, whose total communication complexity is $O(\mathsf{mc}(f))$ field elements. The protocol also consumes $O(\mathsf{mc}(f))$ instances of constant-size OLE correlated randomness,

Statistical Security Loss. The total security loss is $O(\frac{\mathsf{mc}(f)+n^2 \cdot Q_{\mathsf{RO}}}{|\mathbb{F}|})$, where Q_{RO} is the number of random oracle queries that the adversary makes.

7 From Degree 3 to P

In the literature of round-efficient MPC, it is well-known that degree-3 functions are complete for arithmetic $\mathbf{NC^1}$ w.r.t. information-theoretic security, and are complete for **P/poly** w.r.t. com-

putational security, since there exist efficient information-theoretic (resp. computational) non-interactive reductions from **P/poly** (resp. $\mathbf{NC^1}$) to degree-3 functions.

The information-theoretic version of Yao's Garbled circuits [Yao86, IK02] can be viewed as a degree-3 randomized encoding for any boolean circuit. The complexity of information-theoretic Yao is exponential in the circuit depth. Thus the reduction is only efficient for $\mathbf{NC^1}$.

Extension to arithmetic branching programs (ABPs). The technique is extend to arithmetic $\mathbf{NC^1}$ circuits by considering their equivalent modular branching program. Every arithmetic $\mathbf{NC^1}$ circuit has a degree-3 randomized encoding over the same field [IK00, IK02]. Say an arithmetic $\mathbf{NC^1}$ function f can be computed by an arithmetic branching program, given by a matrix M_f where every entry is a linear function in x, such that, for every x, $\det(M_f(x)) = f(x)$. Let ℓ be the size of the matrix. The tasking of computing f can be reduced to computing a degree-3 function $\hat{f}$ who has $O(n^2\ell^{1.5})$ monomials.

By composition, we obtain 2-round MPC protocols for arithmetic $\mathbf{NC^1}$. The protocol uses tensor OLE correlated randomness and RO, and is maliciously secure with output substitution.

Extension to circuits based on black-box PRF. Starting from Yao's garbled circuits, we can get a maliciously secure MPRE for **P/poly** with effective degree 3 that makes black-box use of a PRF F, using the techniques introduced in [DI05, BMR90]. For simplicity, consider garbling a single gate g with input wire u, v and output wire o. For each input/output wire j, each party P_i samples a pair of seeds $s_{j,0}^{(i)}, s_{j,1}^{(i)} \in \mathbb{F}$ corresponding the wire having value 0 or 1; the two labels for wire j is then set to $\boldsymbol{\ell}_{j,x} = s_{j,x}^{(1)} \| \dots \| s_{j,x}^{(n)}$.

To hide the labels of the output wire o, each party locally expands their seeds through F, and hide label $\boldsymbol{\ell}_{o,g(a,b)}$ using the XOR of PRF outputs from all parties. For instance,

$$\boldsymbol{\ell}_{o,g(a,b)} \oplus \mathsf{F}^{\mathrm{MP}}(u, o, d, \boldsymbol{\ell}_{u,a}) \oplus \mathsf{F}^{\mathrm{MP}}(v, o, d', \boldsymbol{\ell}_{u,b})$$

where d, d' are set so that the same PRF output is never reused, and F^{MP} denotes the multi-party version of PRF

$$\mathsf{F}^{\mathrm{MP}}(j, j', d, \boldsymbol{\ell}_{j,x}) := \bigoplus_{i \in [n]} \mathsf{F}(i, j, j', d, \boldsymbol{\ell}_{j,x}[i]) \,.$$

These table entries are further randomly permuted using mask bits k_u, k_v which are additively shared among all parties. The computed encoding is secure as long as one party remains uncorrupted. The computation makes black-box use of the PRF and has effective degree 3 after local evaluation of PRF.

To rigorously present distributed Yao in Fig. 15, we formalize the notations for a boolean circuit. A boolean circuit is specified by a directed acyclic graph $(\mathcal{V}, \mathcal{E})$. Each vertex denotes a wire in the circuit.

- For any $j \in \mathcal{V}$, let x_j denote the wire value of the j-th wire.
- Let $\mathcal{V}_i$ denote the set of the input wires owned by P_i. Let $\mathcal{V}_{\mathsf{in}} := \bigcup_i \mathcal{V}_i$ denote the set of all the input wires.
- Any wire $j \notin \mathcal{V}_{\mathsf{in}}$ is the output of a gate g_j. Let u_j, v_j denotes the input wires of the gate, then $(u_j, j), (v_j, j) \in \mathcal{E}$. In the real execution, $x_j := g_j(x_{u_j}, x_{v_j})$.
- Let $\mathcal{V}_{\mathsf{out}}$ denote the set of all the output wires.

Figure 15: **Distributed Yao as a Malicious MPRE**

Let $\mathbb{F} := \mathsf{GF}(2^\lambda)$.

Input: P_i has $x_j \in \{0,1\}$ for each $j \in \mathcal{V}_i$.

Local Randomness: Party P_i samples a bit $k_j^{(i)} \in \{0,1\}$ for each $j \notin \mathcal{V}_{\text{out}}$, and samples seeds $s_{j,0}^{(i)}, s_{j,1}^{(i)} \in \mathbb{F}$ for each $j \in \mathcal{V}$.

Correlated Randomness: None.

Preprocessing: For each $(u,j) \in \mathcal{E}$, party P_i computes $\hat{\mathbf{s}}_{u,j,x,b}^{(i)} = \mathsf{F}(i,u,j,b,s_{j,x}^{(i)})$ for $x,b \in \{0,1\}$. That is,

$$h_i\Big((x_j,k_j)_{j\in\mathcal{V}_i}, (k_j^{(i)}, s_{j,0}^{(i)}, s_{j,1}^{(i)})_{j\in\mathcal{V}}\Big) := \Big((x_j,k_j)_{j\in\mathcal{V}_i}, (k_j^{(i)}, s_{j,0}^{(i)}, s_{j,1}^{(i)})_{j\in\mathcal{V}}, \\ \big(\mathsf{F}(i,u,j,0,s_{j,0}^{(i)}), \mathsf{F}(i,u,j,1,s_{j,0}^{(i)}), \mathsf{F}(i,u,j,0,s_{j,1}^{(i)}), \mathsf{F}(i,u,j,1,s_{j,1}^{(i)})\big)_{(u,j)\in\mathcal{E}}\Big).$$

Encoding Function: Define mask bit $k_j := \bigoplus_{i\in[n]} k_j^{(i)}$ for each $j \notin \mathcal{V}_{\text{out}}$ and $k_j := 0$ for each $j \in \mathcal{V}_{\text{out}}$. Define label $\boldsymbol{\ell}_{j,x} := s_{j,x}^{(1)} \| \dots \| s_{j,x}^{(n)}$ for each $j \in \mathcal{V}$. Define $\hat{\mathbf{s}}_{u,j,x,b} := \bigoplus_{i\in[n]} \hat{\mathbf{s}}_{u,j,x,b}^{(i)} = \mathsf{F}^{\text{MP}}(u,j,b,\boldsymbol{\ell}_{j,x})$, for each $(u,j) \in \mathcal{E}$.

The encoding consists of three parts

- *Garbled input:* For each $j \in \mathcal{V}_{\text{in}}$, the encoding contains
$$\bar{x}_j := x_j \oplus k_j, \quad \boldsymbol{\ell}_j := \boldsymbol{\ell}_{j,\bar{x}_j}.$$
- *Garbled gate table:* For each $j \notin \mathcal{V}_{\text{in}}$, let u,v be the input wires of g_j, the encoding contains
$$\mathbf{w}_{j,b_1,b_2} := \hat{\mathbf{s}}_{u_j,j,b_1\oplus k_{u_j},b_2\oplus\hat{\mathbf{s}}_{v_j,j,b_2\oplus k_{v_j},b_1} \oplus (k_j \oplus g_j(b_1\oplus k_{u_j}, b_2\oplus k_{v_j}) \| \boldsymbol{\ell}_{j,k_j\oplus g_j(b_1\oplus k_{u_j},b_2\oplus k_{v_j})})$$
for $b_1,b_2 \in \{0,1\}$.
- *Output masks:* For each $j \in \mathcal{V}_{\text{out}}$, the encoding contains k_j.

Decoding Function: For each $j \notin \mathcal{V}_{\text{in}}$, compute
$$\bar{x}_j \| \boldsymbol{\ell}_j = \mathbf{w}_{j,\bar{x}_{u_j},\bar{x}_{v_j}} \oplus \mathsf{F}^{\text{MP}}(u_j,j,\bar{x}_{v_j},\boldsymbol{\ell}_{u_j}) \oplus \mathsf{F}^{\text{MP}}(v_j,j,\bar{x}_{u_j},\boldsymbol{\ell}_{v_j}).$$
Output $\bar{x}_j$ for each $j \in \mathcal{V}_{\text{out}}$. (Note that $k_j = 0$.)

We measure the complexity of the encoding function by number of monomials and degree. The bottleneck is $\boldsymbol{\ell}_{j,k_j\oplus g_j(b_1\oplus k_{u_j},b_2\oplus k_{v_j})}$ with either measure. W.l.o.g., assume g_j is AND gate (i.e., multiplication). Then

$$\boldsymbol{\ell}_{j,k_j\oplus g_j(b_1\oplus k_{u_j},b_2\oplus k_{v_j})} = (\boldsymbol{\ell}_{j,1} - \boldsymbol{\ell}_{j,0})(k_j + (b_1+k_{u_j})(b_2+k_{v_j})) + \boldsymbol{\ell}_{j,0}. \tag{11}$$

Note that $\boldsymbol{\ell}_{j,1}, \boldsymbol{\ell}_{j,0}, k_j, k_{u_j}, k_{v_j}$ are linear, and each of them has $O(n)$ monomials. Thus (11) is of degree 3 and has $O(n^3)$ monomials.

The works of [DI05, BMR90, LPSY15] showed that even if some malicious parties use wrong PRF outputs as their preprocessed inputs, as long as there is one honest party, the privacy of the computed garbled circuit remain.

By composition, we obtain 2-round MPC protocols for **P/poly** that is maliciously secure with output substitution, using $\mathcal{F}_{\mathsf{RO}}$ and tensor OLE correlated randomness.

Remark on boolean inputs. The distributed Yao (Fig. 15) can be viewed as an effective-degree-3 MPRE over a large boolean extension field $\mathbb{F} := \mathsf{GF}(2^\lambda)$. Its inputs and randomness are field elements. While the input bit x_j and the additive share of mask bit $k_j^{(i)}$ are supposed to be chosen

or sampled from $\{0,1\}$. What if a malicious party chooses $x_j \notin \{0,1\}$? There are two answers to this challenge.

- Particularly for distributed Yao, choosing non-boolean input x_j does not give the adversary any advantage.
- Potentially, there may exist a MPRE whose security holds only when a corrupted party chooses boolean inputs (or randomness). In such case, we can enforce the corrupted party to choose boolean input $x \in \{0,1\}$, by relying on our MPC protocol for effective-degree-2 function (Sec. 5). Recall that, our protocol checks if the preprocessed input is well-formed, and aborts the computation otherwise. To ensure $x \in \{0,1\}$, the protocol additionally checks if $x \cdot x = x$. Such check is enabled by the ProveProd command.

8 Lifting Privacy to Security

Sec. 7 constructs 2-round MPC protocol implementing $\mathcal{F}_f^{\mathsf{os}}$ for any function f in **P/poly** (resp. $\mathbf{NC}^1$). In words, it achieves computational (resp. statistical) security with output substitution, against up to $n-1$ malicious corruptions. In this section, we will lift it to security with selective or unanimous abort.

The most natural approach is MAC, which makes the protocol secure with selective abort [IKP10]. Such a protocol is vulnerable to selective abort attack. An adversary can let any subset of honest parties abort, by only substituting the corresponding signatures. To prevent selective abort attacks, we introduce a new tool called *consensus MAC*, in which all honest parties will abort unanimously.

8.1 To Security with Selective Abort via One-Time MAC

A MAC scheme consists of a key generation algorithm KeyGen, a signature algorithm Sign and a verification algorithm Verify. Given a one-time secure MAC scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, as shown by [IKP10], the task of computing f securely with selective abort can be non-interactively reduced to the task of computing the following function securely with output substitution:

$$g((x_1,k_1),\ldots,(x_n,k_n)) = (y, \mathsf{Sign}(y,k_1),\ldots,\mathsf{Sign}(y,k_n)),$$

where $y := f(x_1,\ldots,x_n)$. The reduction is presented in Fig. 16.

Figure 16: Lift security with output substitution to security with selective abort

Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a MAC scheme. Define n-party function g as $g((x_1,k_1),\ldots,(x_n,k_n)) = (y,\sigma_1,\ldots,\sigma_n)$, where $y = f(x_1,\ldots,x_n)$ and $\sigma_i = \mathsf{Sign}(y,k_i)$.

Input: Upon input x_i, party P_i additionally samples $k_i \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Compute: Party P_i inputs x_i, k_i to $\mathcal{F}_g^{\mathsf{os}}$. The functionality sends $y,\sigma_1,\ldots,\sigma_n$ to all parties.

Verify & Output: Party P_i aborts if $\mathsf{Verify}(k_i,y,\sigma_i) \to$ `"reject"`. Otherwise, P_i outputs y.

For arithmetic branching programs (ABPs). W.l.o.g., consider an ABP f that only outputs one number. That is

$$f(x_1,\ldots,x_n) = \det M_f(x_1,\ldots,x_n),$$

where the mapping $(x_1,\ldots,x_n) \mapsto M_f(x_1,\ldots,x_n)$ is affine over a given field $\mathbb{F}$.

To minimize the complexity of the reduction, we use the simplest one-time MAC. The MAC key $k = (a, b)$ consists of two randomly sampled field elements $a, b \in \mathbb{F}$. The signature algorithm is $\mathsf{Sign}(y, k = (a, b)) = ay + b$. Then computing f with selective abort is reduced to computing g with output substitution, where

$$g((x_1, k_1 = (a_1, b_1)), \ldots, (x_n, k_n = (a_n, b_n))) \\ := (f(x_1, \ldots, x_n), a_1 \cdot f(x_1, \ldots, x_n) + b_1, \ldots, a_n \cdot f(x_1, \ldots, x_n) + b_n) .$$

Each ABP in f is transferred into $n + 1$ ABPs. Each of them looks like

$$a_i \cdot f(x_1, \ldots, x_n) + b_i = \det \begin{bmatrix} & & b_i \\ & M_f(x_1, \ldots, x_n) & \\ & & \\ & -1 & a_i \end{bmatrix}$$

which is only $O(1)$ additively larger than f. Overall, the reduction causes an $O(n)$ multiplicative blow-up.

For circuits. For a circuit C, our secure with output substitution protocol computes the distributed Yao of C. An one-time MAC was already embed inside distributed Yao, thus the protocol becomes secure with selective abort for free.

Recall that in distributed Yao (Fig. 15), for each output wire $j \in \mathcal{V}_{\text{out}}$, the decoding function learns the output value x_j together with the corresponding label $\boldsymbol{\ell}_{j,x_j} = s^{(1)}_{j,x_j} \| \ldots \| s^{(n)}_{j,x_j} \in \mathbb{F}^n$.

Currently, the decoding function simply discards the label. A better strategy is to consider $s^{(i)}_{j,x_j}$ as a MAC signature. An honest party will not be fooled because the adversary know nothing about $s^{(i)}_{j,1-x_j}$.

8.2 To Security with Unanimous Abort via an Extra Round

Since we have round-preserving cheap (resp. free) reduction to security with selective abort for ABPs (resp. circuits), we automatically achieves security with unanimous abort with an additional round. In this section, we will show an alternative reduction achieving security with unanimous abort for ABPs, that uses an extra round and and only introduces a constant overhead.

Similarly, we consider an ABP f. In the first two round, compute $y = f(x_1, \ldots, x_n)$ together with $\mathsf{Sign}(y, k)$, where Sign is one-time MAC and the MAC key k is additively shared among all parties. In the last round, open the MAC key so that everyone can verify. This reduction is presented in Fig. 17.

Lemma 8.1. *The protocol in Fig. 17 implements* $\mathcal{F}_f^{\mathsf{abort}}$.

Proof. Since $k := k_1 + \cdots + k_n$ is hidden when the adversary chooses substitution output $(y', \sigma') \neq (y, \sigma)$, the verification $\mathsf{Verify}(k, y, \sigma)$ rejects with overwhelming probability. □

By instantiating the MAC scheme by one-time MAC, the task of computing f with unanimous abort is reduced to computing g with output substitution (using one less round), where

$$g((x_1, k_1 = (a_1, b_1)), \ldots, (x_n, k_n = (a_n, b_n))) := \left(f(x_1, \ldots, x_n), (\textstyle\sum_i a_i) \cdot f(x_1, \ldots, x_n) + (\sum_i b_i)\right) .$$

Figure 17: Lift security with output substitution to security with unanimous abort

Defines n-party function g such that: $g((x_1, k_1), \ldots, (x_n, k_n)) = (y, \sigma)$ where $y = f(x_1, \ldots, x_n)$ and $\sigma = \mathsf{Sign}(y, k_1 + \cdots + k_n)$.

Input: Upon input x_i, party P_i additionally samples $k_i \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Compute: Party P_i inputs x_i, k_i to $\mathcal{F}_g^{\mathsf{os}}$. The functionality sends y, σ to all parties.

Commit: In parallel to the computation of g, party P_i publicly commits to k_i.

Open: After the computation of g, party P_i opens k_i.

Verify & Output: Party P_i aborts if $\mathsf{Verify}(k_1 + \cdots + k_n, y, \sigma_i) \to$ `"reject"`. Otherwise, P_i outputs y.

g's second output term looks like

$$\sum_i a_i \cdot f(x_1, \ldots, x_n) + \sum_i b_i = \det \begin{bmatrix} & & \sum_i b_i \\ & M_f(x_1, \ldots, x_n) & \\ & & \\ & -1 & \sum_i a_i \end{bmatrix},$$

which is as complex as f (assuming w.l.o.g. that M_f has at least n monomials). Thus g only has a constant overhead compared with f.

8.3 To Security with Unanimous Abort via Consensus MAC

Consensus MAC[9] is designed for lifting security with output substitution. It allow signing a message using a collection of keys $k_1, \ldots, k_n$, such that a forged signature conflicts every single key k_i with high probability. For the purpose of constructing MPC, public key is unnecessary, and one-time security is sufficient.

Definition 7 ([ACGJ19])**.** A consensus MAC scheme is a tuple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ of three polynomial-time algorithms

- KeyGen is a randomized algorithm, on input 1^λ, outputs a key $k \in \mathcal{K}$, where $\mathcal{K}$ denotes the key space.
- Sign is a deterministic algorithm, on input a message $m \in \mathcal{M}$ and keys $k_1, \ldots, k_n$, output a signature σ. Here $\mathcal{M}$ denotes the message space.
- Verify on input a key k, a message m, a signature σ, output `"accept"` or `"reject"`.

that satisfies correctness and one-time consensus security.
Correctness: For any message m, sample keys $k_i \leftarrow \mathrm{KeyGen}(1^\lambda)$, and compute the signature $\sigma \leftarrow \mathrm{Sign}(m, k_1, \ldots, k_n)$, then

$$\forall i \in [n], \mathrm{Verify}(k_i, m, \sigma) \to \texttt{"accept"}$$

with overwhelming probability.
One-time Consensus Security: Any polynomial-time adversary wins the following security game with negligible probability.

[9] The consensus MAC is the *One-Time Multi-Key MAC* introduced in [ACGJ19].

- The adversary chooses a message m, a set of corrupted parties $C \subseteq [n]$, keys k_i for each corrupted $i \in C$, and sends them to the challenger.
- The challenger samples $k_i \leftarrow \mathrm{KeyGen}(1^\lambda)$ for each honest $i \notin C$, computes the signature $\sigma \leftarrow \mathrm{Sign}(m, k_1, \ldots, k_n)$, and sends σ to the adversary.
- The adversary outputs m', σ'.
- The adversary wins if $(m', \sigma') \neq (m, \sigma)$, and $\mathrm{Verify}(k_i, m', \sigma') \to$ `"accept"` for some $i \notin C$.

An example of consensus MAC scheme that only signs one field element is presented in Sec. 2.4. Here we present another consensus MAC scheme (Fig. 18) that supports long message. In the scheme, i-th party's key consists of a field element a_i and a function h_i that maps the message to a field element. As we are going to show, it suffices to sample h_i from a 2-universal hash function family. To sign a message m, first compute $b_i \leftarrow h_i(m)$ for all i, the signature σ is the degree-$(n-1)$ polynomial that $\sigma(a_i) = b_i$.

Say the adversary is given message m and signature σ, and want to generate $(m', \sigma') \neq (m, \sigma)$ such that the i-th party will accept (m', σ') with good probability. The i-th party computes $b' \leftarrow h_i(m')$ and check if $\sigma'(a_i) = b'$.

- If $m' \neq m$, since the adversary has no information about b', the check fails with overwhelming probability.
- If $m' = m$, then $b' = \sigma(a_i)$. That is, the check passes if and only if $\sigma'(a_i) = \sigma(a_i)$. The two polynomial σ', σ agree on at most $n-1$ points. Since the adversary has almost no knowledge about a_i, with overwhelming probability, a_i is not among the points where σ', σ agree.

As every honest party will abort with overwhelming probability, the honest parties will abort unanimously.

Figure 18: Consensus MAC Scheme

Initialize: On input 1^λ, decide a field $\mathbb{F}$ such that $|\mathbb{F}| \geq 2^\lambda$.

KeyGen: Sample a random $a \in \mathbb{F}$, and sample $h : \mathcal{M} \to \mathbb{F}$ from a 2-universal hash function family (e.g., a random affine function). Output $k = (a, h)$.

Sign: On input a message $m \in \mathcal{M}$ and keys $k_1, \ldots, k_n$. Compute $b_i \leftarrow h_i(m)$. If there exist distinct i, j such that $a_i = a_j$, output $\perp$. Otherwise, compute the unique degree-$(n-1)$ polynomial σ such that $\sigma(a_i) = b_i$ for all $i \in [n]$, and output σ.

Verify: On input a key $k = (a, h)$, a message m, a signature σ. Compute $b \leftarrow k(m)$. Output `"accept"` if $\sigma(a) = b$.

Lemma 8.2. *The scheme in Fig. 18 is a secure consensus MAC scheme.*

Proof. The correctness is straight-forward. Sign only outputs $\perp$ upon a collision, which happens with probability $n^2/|\mathbb{F}|$ if the keys are randomly sampled.

For security, consider any adversary. As defined in the security game:

- The adversary chooses a message m, a set of corrupted parties $C \subseteq [n]$, keys $k_i = (a_i, h_i)$ for each corrupted $i \in C$, and sends them to the challenger.

- The challenger samples $k_i = (a_i, h_i)$ for each honest $i \notin C$, computes $\sigma \leftarrow \mathrm{Sign}(m, k_1, \ldots, k_n)$, and sends σ to the adversary.
- The adversary outputs m', σ'.

The adversary's winning probability is negligible:

- Case I, $m' = m, \sigma' = \sigma$: In such case, by definition, the adversary never wins.
- Case II, $m' = m, \sigma = \perp$: For each honest party P_i, denote $b_i \leftarrow h_i(m)$. In such case, the adversary learns no information about b_i for every $i \notin C$. Therefore, the probability an honest P_i accepts the signature σ' is no more than $1/|\mathbb{F}|$.
- Case III, $m' = m, \sigma' \neq \sigma$: For each honest party P_i, denote $b_i \leftarrow h_i(m)$. An honest P_i accepts only if $\sigma'(a_i) = b_i = \sigma(a_i)$. For each $i \notin C$, the adversary knows no information about a_i other than that $a_i \notin \{a_j\}_{j \in C}$. The two polynomials σ', σ agree on at most $n-1$ points. Due to the randomness of a_i conditioning on the adversary's view,

$$\Pr[\sigma'(a_i) = \sigma(a_i)] \leq \frac{n-1}{|\mathbb{F}| - |C|}.$$

- Case IV, $m' \neq m$: For each honest party P_i, denote $b'_i \leftarrow h_i(m')$. Due to the 2-universal hashing h_i, the adversary knows no information about b'_i for every $i \notin C$. Therefore, the probability an honest P_i accepts the signature σ' is no more than $1/|\mathbb{F}|$. □

Given consensus MAC scheme, lifting security with output substitution to security with (unanimous) abort becomes easy. Every party P_i additionally sample a signature key k_i. They jointly compute $y = f(x_1, \ldots, x_n)$ together with the signature, using the secure with output substitution protocol for **P/poly**. Then every party can verify the output. By the security of consensus MAC, all honest parties will abort unanimously if the adversary substitute the output and/or the signature.

Figure 19: Lift security with output substitution to security with unanimous abort

Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a consensus MAC scheme. Defines n-party function g such that: the i-th input of g consists of f's i-th input and a MAC key k_i; the output of g consists of $y = f(x_1, \ldots, x_n)$ and $\sigma = \mathsf{Sign}(y, k_1, \ldots, k_n)$.

Input: Upon input x_i, party P_i additionally samples $k_i \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Compute: Party P_i inputs x_i, k_i to $\mathcal{F}_g^{\mathsf{os}}$. The functionality sends y, σ to all parties.

Verify & Output: Party P_i aborts if $\mathsf{Verify}(k_i, y, \sigma) \to$ `"reject"`. Otherwise, P_i outputs y.

Lemma 8.3. *The protocol in Fig. 19 implements $\mathcal{F}_f^{\mathsf{abort}}$.*

Proof. Every honest party will abort with overwhelming probability if the adversary replace the output by $(y', \sigma') \neq (y, \sigma)$. This is guaranteed by the consensus MAC scheme. □

9 Putting Pieces Together

In this section, we put components built in previous sections together to form the full MPC protocols for computing Boolean circuits, and describe its complexity. We also analyze the complexity

of the protocols for computing ABPs. Here, the *complexity* of a MPC protocol refers to the number of field elements in the communication, and the number of constant-size OLE correlated randomness used.

Recall that in Sec. 6, we showed that for any degree-3 function f, there is a 2-round MPC protocol that is maliciously secure with output substitution. The complexity of the protocol is $O(\mathsf{mc}(f))$, where $\mathsf{mc}(f)$ denotes the total number of monomials in f. The security level of the protocol is $O(\frac{\mathsf{mc}(f)+n^2\cdot Q_{\mathsf{RO}}}{|\mathbb{F}|})$, where Q_{RO} is the number of random oracle queries that the adversary makes.

MPC protocols for circuits. Let f be a function that is computed by a Boolean circuit of ℓ gates. By composing with the reduction from **P/poly** to degree-3 in Sec. 7, the tasking of computing f can be reduced to computing a degree-3 function $\hat{f}$ such that $\mathsf{mc}(\hat{f}) = O(n^3\ell)$. Therefore, there is a 2-round MPC protocol that is maliciously secure with output substitution. The complexity of the protocol is $O(n^3\ell)$. The protocol makes black-bow calls to PRF. If the PRF function is also modeled as a random oracle, the statistical security level is $O(\frac{n^3\ell+n^2\cdot Q_{\mathsf{RO}}}{|\mathbb{F}|})$.

Combining with the lifting in Sec. 8, there are, as shown in Fig. 20,

- A 3-round MPC protocol that is maliciously secure with unanimous abort and has the same complexity.
- A 2-round MPC protocol that is maliciously secure with selective abort and has the same complexity.
- A 2-round MPC protocol that is maliciously secure with unanimous abort and has an additive polynomial growth on the complexity.

	ℓ-size ABP		ℓ-gate circuit
	complexity	statistical security	complexity
2-round, security w/ output substitution	$O(n^2\ell^{1.5})$	$O(\frac{n^2\ell^{1.5}+n^2\cdot Q_{\mathsf{RO}}}{\lvert\mathbb{F}\rvert})$	$O(n^3\ell)$
3-round, security w/ unanimous abort			
2-round, security w/ selective abort	$O(n^3\ell^{1.5})$	$O(\frac{n^3\ell^{1.5}+n^2\cdot Q_{\mathsf{RO}}}{\lvert\mathbb{F}\rvert})$	
2-round, security w/ unanimous abort	$\mathrm{poly}(n,\ell)$	$O(\frac{\mathrm{poly}(n,\ell)+n^2\cdot Q_{\mathsf{RO}}}{\lvert\mathbb{F}\rvert})$	$O(n^3\ell)+\mathrm{poly}(n,\lambda)$

Complexity is measure by the number of field elements communicated.

Figure 20: The final MPC protocols

The full protocols for Boolean circuits For completeness, we unfold the 2-round MPC protocol secure with unanimous abort for circuits below. Let $f \in$ **P/poly** be the function to evaluate.

Input and correlated randomness Party P_i receives input x_i and its portion of correlated randomness r'_i. The correctness randomness contains two parts $r'_i = (r'^{\mathrm{MPRE}}_i, r'^{\mathrm{MPC}}_i)$, the former will be consumed by our effective-degree-2 MPRE and the latter will be consumed by our 2-round MPC for computing effective-degree-2 functions.

Authenticate the output Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be the consensus MAC scheme constructed in Sec. 8. Define f^{auth} as

$$f^{\text{auth}}((x_1, k_1), \ldots, (x_n, k_n)) = \Big(f(x_1, \ldots, x_n), \mathsf{Sign}(f(x_1, \ldots, x_n), k_1, \ldots, k_n)\Big)$$

Party P_i samples key $k_i \leftarrow \mathsf{KeyGen}$. And let $x_i' := (x_i, k_i)$. This step corresponds to the reduction in Sec. 8 that reduces the task of securely computing f with unanimous abort to securely computing f^{auth} with output substitution.

Distributed Yao Let $f^{\text{dYao}}, h_1^{\text{dYao}}, \ldots, h_n^{\text{dYao}}, \mathsf{Dec}^{\text{dYao}}$ be the distributed-Yao MPRE for f^{auth}. Party P_i samples the randomness r_i^{dYao} and computes $x_i'' = h_i^{\text{dYao}}(x_i', r_i^{\text{dYao}})$, where the preprocessing h_i^{dYao} evaluates a PRF function.

This step corresponds to the reduction in Sec. 7 that reduces securely computing f^{auth} (with output substitution) to securely computing the degree-3 function f^{dYao} (with output substitution).

Canonicalization of degree-3 function As shown by [BGI+18, GIS18, LLW20], there is a MPRE $f^{\text{can}}, h_1^{\text{can}}, \ldots, h_n^{\text{can}}, \mathsf{Dec}^{\text{can}}$ for f^{dYao}, whose encoding function f^{can} is a canonical degree-3 function as defined in Sec. 4. Party P_i samples the randomness r_i^{can}, and computes $x_i''' = h_i^{\text{can}}(x_i'', r_i^{\text{can}}) = (x_i'', r_i^{\text{can}})$.

Effective-degree-2 MPRE for canonical degree-3 functions Let $\hat{f}, h_1, \ldots, h_n, \mathsf{Dec}$ be the semi-malicious effective-degree-2 MPRE for f^{can}, constructed in Sec. 4. Party P_i samples local randomness r_i^{MPRE} and holds correlated randomness $r_i'^{\text{MPRE}}$.

This step corresponds to the reduction in Sec. 4 that reduces securely computing f^{can} to securely computing an effective degree-2 function $\hat{f} \circ h$.

2-round MPC for effective-degree-2 functions Use the MPC protocols for computing effective-degree-2 functions constructed in Sec. 5 to evaluate

$$\hat{y} = \hat{f}(h_i(x_1''', r_1^{\text{MPRE}}, r_1'^{\text{MPRE}}), \ldots, h_n(x_n''', r_n^{\text{MPRE}}, r_n'^{\text{MPRE}})) \ .$$

In the protocol execution, P_i uses input $(x_i''', r_i^{\text{MPRE}}, r_i'^{\text{MPRE}})$, and correlated randomness $r_i'^{\text{MPC}}$. At the end of the protocol execution, P_i obtains $\hat{y}$.

Decode and verify Party P_i decodes $\hat{y}$

$$(y, \sigma) \leftarrow \mathsf{Dec}^{\text{dYao}}(\mathsf{Dec}^{\text{can}}(\mathsf{Dec}(\hat{y}))) \ ,$$

It aborts if $\mathsf{Verify}(k_i, y, \sigma) \to$ `"reject"`, otherwise outputs y.

MPC for arithmetic branching programs. Let f be an arithmetic $\mathbf{NC^1}$ function that is computed by an arithmetic branching program of size ℓ, where ℓ is the size of the matrix M_f such that $\det M_f(x) = f(x)$. By composing with the reduction from $\mathbf{NC^1}$ to degree-3 in Sec. 7, the tasking of computing f can be reduced to computing a degree-3 function $\hat{f}$ such that $\mathsf{mc}(\hat{f}) = O(n^2\ell^{1.5})$. Therefore, there is a 2-round MPC protocol that is maliciously secure with output substitution. The complexity of the protocol is $O(n^2\ell^{1.5})$ and the security level is $O(\frac{n^2\ell^{1.5}+n^2 \cdot Q_{\mathsf{RO}}}{|\mathbb{F}|})$.

Combining with the lifting in Sec. 8, there are, as shown in Fig. 20,

- A 3-round MPC protocol that is maliciously secure with unanimous abort and has the same complexity and security level.

- A 2-round MPC protocol that is maliciously secure with selective abort and has an $O(n)$ blow-up on the complexity and security level.
- A 2-round MPC protocol that is maliciously secure with unanimous abort and has a polynomial blow-up on the complexity and security level.

The full protocol for computing ABPs is the same as the above protocol for Boolean circuits, except that distributed-Yao is replaced with distributed-AIK.

Acknowledgement

We thank Antigoni Polychroniadou for being our shepherd and her help, and the anonymous Crypto reviewers for their helpful comments and suggestions. We thank Hoeteck Wee for being part of the initial discussion and his suggestion of directions. Finally, we thank Stefano Tessaro for his comments.

Huijia Lin and Tianren Liu were supported by NSF grants CNS-1936825 (CAREER), CNS-2026774, a JP Morgan AI research Award, a Cisco research award, and a Simons Collaboration on the Theory of Algorithmic Fairness. In addition, Tianren Liu was supported by NSFC excellent young scientists fund program.

References

[ABT18] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect secure computation in two rounds. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 152–174. Springer, Heidelberg, November 2018.

[ABT19] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Degree 2 is complete for the round-complexity of malicious MPC. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 504–531. Springer, Heidelberg, May 2019.

[ACGJ19] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 532–561. Springer, Heidelberg, May 2019.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Heidelberg, November 2020.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Heidelberg, August 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BDK+11] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2011.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Heidelberg, April / May 2017.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

[BGMM20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round MPC from DDH. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 320–348. Springer, Heidelberg, November 2020.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018.

[BLPV18] Fabrice Benhamouda, Huijia Lin, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Two-round adaptively secure multiparty computation from

standard assumptions. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 175–205. Springer, Heidelberg, November 2018.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CDI+19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 462–488. Springer, Heidelberg, August 2019.

[CGP15] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 557–585. Springer, Heidelberg, March 2015.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, August 2005.

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 586–613. Springer, Heidelberg, March 2015.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Heidelberg, February 2014.

[Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Heidelberg, August 1999.

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 123–151. Springer, Heidelberg, November 2018.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, Heidelberg, August 2015.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185. Springer, Heidelberg, August 1987.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 614–637. Springer, Heidelberg, March 2015.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th FOCS*, pages 588–599. IEEE Computer Society Press, October 2017.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 174–184. IEEE Computer Society, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP 2002*, volume 2380 of *LNCS*, pages 244–256. Springer, Heidelberg, July 2002.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 577–594. Springer, Heidelberg, August 2010.

[IKSS21] Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. On the round complexity of black-box secure MPC. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 214–243, Virtual Event, August 2021. Springer, Heidelberg.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[LLW20] Huijia Lin, Tianren Liu, and Hoeteck Wee. Information-theoretic 2-round MPC without round collapsing: Adaptive security, and more. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 502–531. Springer, Heidelberg, November 2020.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, Heidelberg, August 2015.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[Pas12] Anat Paskin-Cherniavsky. *Secure computation with minimal interaction*. PhD thesis, Computer Science Department, Technion, Haifa, Israel, 2012. advised by Yuval Ishai and Eyal Kushilevitz.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Heidelberg, October / November 2016.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Implementation of OLE from OT

In this section, we present a protocol for generation of tensor OLE correlated randomness (Fig. 21). The protocol is secure against malicious adversary, and is based on OT and commitment.

We first present a semi-honest protocol in Sec. A.1 using OT. The security is lifted to malicious security in Sec. A.2. The lift causes a constant-factor overhead on the communication complexity.

The dimension of tensor OLE correlated randomness is symmetric. A pair of parties get $\mathbf{a}, \mathbf{b} \in \mathbb{F}^k$ respectively and the additive sharing of $\mathbf{a}\mathbf{b}^\intercal$. It is easy to generalized the problem so that $\mathbf{a}, \mathbf{b}$ may have different lengths.

We let the field $\mathbb{F}$ be $\mathsf{GF}(2^\lambda)$, where λ is the security parameter. It is easy to generalized the protocol for any boolean extension field.

Figure 21: The TOLE Correlation Functionality $\mathcal{F}_{\mathsf{OLEcor}}$

Parameter: A finite field $\mathbb{F}$ and the number of parties n. A subset $C \subseteq [n]$ of parties are corrupted by the adversary.

Generate: Upon P_i sends $(\texttt{"Gen"}, \mathsf{sid} = (i, j, k, \mathsf{seqnum}))$ and P_j sends $(\texttt{"Gen"}, \mathsf{sid} = (i, j, k, \mathsf{seqnum}))$, where session id sid consists of two participants P_i, P_j, the dimension $k_i \times k_j$, and a sequence number seqnum.

The functionality samples random $\mathbf{a}_i \in \mathbb{F}^k, \mathbf{a}_j \in \mathbb{F}^k, \mathbf{B}_i \in \mathbb{F}^{k\times k}, \mathbf{B}_j \in \mathbb{F}^{k\times k}$, such that $\mathbf{a}_i\mathbf{a}_j^\intercal = \mathbf{B}_i + \mathbf{B}_j^\intercal$, and stores $(\mathsf{sid}, \mathbf{a}_i, \mathbf{a}_j, \mathbf{B}_i, \mathbf{B}_j)$ if no entry was previous stored under session id sid.

If $i \in C$, the functionality sends $(\mathsf{sid}, \mathbf{a}_i, \mathbf{B}_i)$ to the adversary. If $j \in C$, the functionality sends $(\mathsf{sid}, \mathbf{a}_j, \mathbf{B}_j)$ to the adversary. In either case, the functionality waits the adversary to abort the protocol.

If the adversary does not abort the protocol, the functionality sends $(\mathsf{sid}, \mathbf{a}_i, \mathbf{B}_i)$ to P_i, sends $(\mathsf{sid}, \mathbf{a}_j, \mathbf{B}_j)$ to P_j.

A.1 Semi-honest Tensor-OLE Correlation Generation

Alice samples random $\mathbf{a} \in \mathbb{F}^{k_1}$, $\mathbf{C} \in \mathbb{F}^{k_1\times k_2}$. Bob samples random $\mathbf{b} \in \mathbb{F}^{k_2}$. It suffices to let Bob learn $\mathbf{D} = \mathbf{a}\mathbf{b}^\intercal - \mathbf{C}$.

Note that, the mapping from $\mathbf{b}$ to $\mathbf{D} = \mathbf{a}\mathbf{b}^\intercal - \mathbf{C}$ affine over $\mathsf{GF}(2)$. Let $(\mathbf{a})_{(2)} \in \mathsf{GF}(2)^{k_1\lambda}$ denotes the bit representation of $\mathbf{a}$, let $(\mathbf{C})_{(2)} \in \mathsf{GF}(2)^{k_1k_2\lambda}$ denotes the bit representation of $\mathbf{C}$. Similarly define $(\mathbf{b})_{(2)}$ and $(\mathbf{D})_{(2)}$. Then $\mathbf{D} = \mathbf{a}\mathbf{b}^\intercal - \mathbf{C}$ if and only if

$$(\mathbf{D})_{(2)} = M_{\mathbf{a}} \cdot (\mathbf{b})_{(2)} - (\mathbf{C})_{(2)} \,, \tag{12}$$

where $M_{\mathbf{a}} \in \mathsf{GF}(2)^{k_1k_2\lambda\times k_2\lambda}$ is a bit matrix determined by $\mathbf{a}$. Therefore, using Oblivious Transfer, Alice inputs $M_{\mathbf{a}}, \mathbf{C}$, Bob inputs $\mathbf{b}$, then Bob learns $\mathbf{D}$.

Such protocol is not maliciously secure, because corrupted Alice might input a maliciously generated matrix instead of $M_{\mathbf{a}}$.

A.2 Maliciously Secure Implement

The protocol is sketched in Sec. 2.5. Alice samples $\mathbf{a}, \mathbf{C}_1$. Bob samples $\mathbf{b}, \mathbf{D}_2$. Alice and Bob learn $\mathbf{C}_2, \mathbf{D}_1$ respectively from $\mathcal{F}_{\mathsf{OT}}$ such that

$$(\mathbf{D}_1)_{(2)} = M_{\mathbf{a}} \cdot (\mathbf{b})_{(2)} - (\mathbf{C}_1)_{(2)} \,, \quad (\mathbf{C}_2)_{(2)} = M_{\mathbf{b}} \cdot (\mathbf{a})_{(2)} - (\mathbf{D}_2)_{(2)} \,.$$

Say Alice is corrupted. She inputs M^* into $\mathcal{F}_{\mathsf{OT}}$ instead, so that Bob actually learns $\mathbf{D}_1$ that

$$(\mathbf{D}_1)_{(2)} = M^* \cdot (\mathbf{b})_{(2)} - (\mathbf{C}_1)_{(2)}.$$

To detect if Alice is misbehaving, Bob samples random $H \in \mathsf{GF}(2)^{\lambda \times \ell_1 \ell_2 \lambda}$, challenges Alice with H. On receiving Alice's answer $\mathbf{v}$, Bob checks if $\mathbf{v} = H \cdot (\mathbf{D}_1 - \mathbf{D}_2)_{(2)}$. Say $M^* = M_{\mathbf{a}} + \Delta$, then

$$\begin{aligned}(\mathbf{D}_1 - \mathbf{D}_2)_{(2)} &= (M_{\mathbf{a}} + \Delta) \cdot (\mathbf{b})_{(2)} - (\mathbf{C}_1)_{(2)} - \left(M_{\mathbf{b}} \cdot (\mathbf{a})_{(2)} - (\mathbf{C}_2)_{(2)}\right) \\ &= \Delta \cdot (\mathbf{b})_{(2)} - (\mathbf{C}_1 - \mathbf{C}_2)_{(2)}. \qquad (13)\end{aligned}$$

$$H \cdot (\mathbf{D}_1 - \mathbf{D}_2)_{(2)} = H \cdot \Delta \cdot (\mathbf{b})_{(2)} - H \cdot (\mathbf{C}_1 - \mathbf{C}_2)_{(2)}. \qquad (14)$$

Alice meets Bob's challenge if and only if she guesses $H \cdot \Delta \cdot (\mathbf{b})_{(2)}$ correctly. Since $\mathbf{b}$ is completely random and is hidden from Alice, she succeeds with probability $2^{-\operatorname{rank}(H\Delta)}$. Her chance depends on the rank of Δ,

- If Δ has high rank, say, $\operatorname{rank}(\Delta) \geq \lambda$, then with overwhelming probability, $H\Delta$ also has high rank. So Alice will be caught with overwhelming probability.
- If Δ has low rank, say $\operatorname{rank}(\Delta) < \lambda$, then Alice might be lucky enough to guess $H \cdot \Delta \cdot (\mathbf{b})_{(2)}$.
- Note that, if Alice is honest, then Δ is the zero matrix. In such case, she can always guess $H \cdot \Delta \cdot (\mathbf{b})_{(2)}$ correctly.

If $\operatorname{rank}(\Delta) < \lambda$, Alice has a non-negligible of guessing $H \cdot \Delta \cdot (\mathbf{b})_{(2)}$ correctly. Moreover, if Alice guesses it correctly, she learns $H \cdot \Delta \cdot (\mathbf{b})_{(2)}$ as a leakage.

If Bob is corrupted, it still holds that $\mathbf{a}\mathbf{b}^\intercal = \mathbf{C}_1 + \mathbf{D}_1$. Corrupted Bob may input $M^{**} = M_{\mathbf{b}} + \Delta'$ into $\mathcal{F}_{\mathsf{OT}}$ so that Alice learns $\mathbf{C}_2$ that

$$(\mathbf{C}_2)_{(2)} = M^{**} \cdot (\mathbf{a})_{(2)} - (\mathbf{D}_2)_{(2)} \, .$$

Later, the honest Alice will answer Bob's challenge with $\mathbf{v} := H \cdot (\mathbf{C}_1 - \mathbf{C}_2)_{(2)}$. Since

$$\mathbf{v} := H \cdot (\mathbf{C}_1 - \mathbf{C}_2)_{(2)} = -H\Delta' \cdot (\mathbf{a})_{(2)} - H \cdot (\mathbf{D}_1 - \mathbf{D}_2)_{(2)} \, ,$$

Bob will learn $H\Delta' \cdot (\mathbf{a})_{(2)}$ as a leakage of $\mathbf{a}$.

We use extraction to remove any leakage. Alice samples a random matrix $\mathbf{E}$ and computes $\mathbf{E} \cdot \mathbf{a}$. By Leftover Hash Lemma, if the random matrix $\mathbf{E}$ is of proper dimension, $\mathbf{E} \cdot \mathbf{a}$ is independent from the leakage $H\Delta' \cdot (\mathbf{a})_{(2)}$, for most choice of $\mathbf{E}$.

Symmetrically, Bob samples a random matrix $\mathbf{F}$ can computes $\mathbf{F} \cdot \mathbf{b}$. By Leftover Hash Lemma, $\mathbf{F} \cdot \mathbf{b}$ is independent from the leakage $H \cdot \Delta \cdot (\mathbf{b})_{(2)}$.

Alice and Bob exchange $\mathbf{E}, \mathbf{F}$, and compute $(\mathbf{a}', \mathbf{C}'), (\mathbf{b}', \mathbf{D}')$ respectively

$$\mathbf{a}' = \mathbf{E} \cdot \mathbf{a} \, , \quad \mathbf{C}' = \mathbf{E} \cdot \mathbf{C}_1 \cdot \mathbf{F}^\intercal \, , \quad \mathbf{b}' = \mathbf{F} \cdot \mathbf{b} \, , \quad \mathbf{D}' = \mathbf{E} \cdot \mathbf{D}_1 \cdot \mathbf{F}^\intercal \, .$$

It is easy to verify that $\mathbf{a}'\mathbf{b}'^\intercal = \mathbf{C}' + \mathbf{D}'$.

Up to this point, Alice and Bob jointly generate the arithmetic analog of random OT. It satisfies the first of the following properties but not the second:

- ($\mathbf{a}'$ is hidden from Bob.) If Alice is honest, $\mathbf{a}'$ is uniformly random conditioning on Bob's knowledge.
- ($\mathbf{a}'$ is not chosen by Alice.) If Bob is honest, $\mathbf{a}'$ is uniformly random.

For many purposes (e.g. the commit-and-prove-linear mechanism in Fig. 5), the first property is already sufficient. But lacking the second property is fatal for our protocol (). To enforce uniform randomness, Alice and Bob have to jointly toss the coins.

More precisely, Alice samples a random vector $\mathbf{s}$ and sends it to Bob, then Bob computes $\mathbf{b}'' = \mathbf{b}' + \mathbf{s}$. As long as one of Alice and Bob is honest, the distribution of $\mathbf{b}''$ will be uniform. Similarly, Bob samples $\mathbf{r}, \mathbf{R}$ at random and sends them to Alice. They compute $(\mathbf{a}'', \mathbf{C}''), (\mathbf{b}'', \mathbf{D}'')$ respectively

$$\mathbf{a}'' = \mathbf{a}' + \mathbf{r}\,, \quad \mathbf{C}'' = \mathbf{a}''\mathbf{s}^\intercal + \mathbf{C}' + \mathbf{R}\,, \quad \mathbf{b}'' = \mathbf{b}' + \mathbf{s}\,, \quad \mathbf{D}'' = \mathbf{D}' + \mathbf{r}\mathbf{b}'^\intercal - \mathbf{R}\,,$$

so that

$$\mathbf{a}''\mathbf{b}''^\intercal = \mathbf{a}''\mathbf{s}^\intercal + \mathbf{a}'\mathbf{b}'^\intercal + \mathbf{r}\mathbf{b}'^\intercal = \underbrace{\mathbf{a}''\mathbf{s}^\intercal + \mathbf{C}' + \mathbf{R}}_{=\mathbf{C}''} + \underbrace{\mathbf{D}' + \mathbf{r}\mathbf{b}'^\intercal - \mathbf{R}}_{=\mathbf{D}''}.$$

To prevent corrupted Alice (resp. Bob) to choose $\mathbf{s}$ (resp. $\mathbf{r}, \mathbf{R}$) adaptively, $\mathbf{s}$ (resp. $\mathbf{r}, \mathbf{R}$) should be committed by the beginning.

Figure 22: Tensor-OLE Correlated Randomness Generation Protocol

Generate Alice samples $\mathbf{a} \leftarrow \mathbb{F}^{\ell_1}$, $\mathbf{C}_1 \leftarrow \mathbb{F}^{\ell_1 \times \ell_2}$. Bob samples $\mathbf{b} \leftarrow \mathbb{F}^{\ell_2}$, $\mathbf{D}_2 \leftarrow \mathbb{F}^{\ell_1 \times \ell_2}$.

Alice computes $M_{\mathbf{a}} \in \mathsf{GF}(2)^{\ell_1\ell_2\lambda \times \ell_2\lambda}$, lets Bob learn $\mathbf{D}_1$ from $\mathcal{F}_{\mathsf{OT}}$ such that

$$(\mathbf{D}_1)_{(2)} := M_{\mathbf{a}} \cdot (\mathbf{b})_{(2)} - (\mathbf{C}_1)_{(2)}.$$

Bob computes $M_{\mathbf{b}} \in \mathsf{GF}(2)^{\ell_1\ell_2\lambda \times \ell_1\lambda}$, lets Alice learn $\mathbf{C}_2$ from $\mathcal{F}_{\mathsf{OT}}$ such that

$$(\mathbf{C}_2)_{(2)} := M_{\mathbf{b}} \cdot (\mathbf{a})_{(2)} - (\mathbf{D}_2)_{(2)}.$$

Test Bob samples $H \leftarrow \mathsf{GF}(2)^{2\lambda \times \ell_1\ell_2\lambda}$ and sends H. Alice computes

$$\mathbf{v} := -H \cdot (\mathbf{C}_1 - \mathbf{C}_2)_{(2)}\,,$$

and sends $\mathbf{v}$ to Bob. Bob aborts if $\mathbf{v} \neq H \cdot (\mathbf{D}_1 - \mathbf{D}_2)_{(2)}$.

Extract Alice (resp. Bob) samples and commits $\mathbf{E} \leftarrow \mathbb{F}^{k_1 \times \ell_1}$ (resp. $\mathbf{F} \leftarrow \mathbb{F}^{k_2 \times \ell_2}$) at the beginning of the protocol. They open $\mathbf{E}, \mathbf{F}$ to each other. Alice computes $\mathbf{a}', \mathbf{C}'$, Bob computes $\mathbf{b}', \mathbf{D}'$, such that

$$\mathbf{a}' := \mathbf{E} \cdot \mathbf{a}\,, \qquad \mathbf{b}' := \mathbf{F} \cdot \mathbf{b}\,,$$
$$\mathbf{C}' := \mathbf{E} \cdot \mathbf{C}_1 \cdot \mathbf{F}^\intercal\,, \quad \mathbf{D}' := \mathbf{E} \cdot \mathbf{D}_1 \cdot \mathbf{F}^\intercal\,.$$

Randomize Alice samples and commits $\mathbf{s} \leftarrow \mathbb{F}^{k_2}$ at the beginning of the protocol. Bob samples and commits $\mathbf{r} \leftarrow \mathbb{F}^{k_1}$, $\mathbf{R} \leftarrow \mathbb{F}^{k_1 \times k_2}$ at the beginning of the protocol. They open $\mathbf{s}, \mathbf{r}, \mathbf{R}$. Alice computes $\mathbf{a}'', \mathbf{C}''$, Bob computes $\mathbf{b}'', \mathbf{D}''$, such that

$$\mathbf{a}'' := \mathbf{a}' + \mathbf{r}\,, \qquad \mathbf{b}'' := \mathbf{b}' + \mathbf{s}\,,$$
$$\mathbf{C}'' := \mathbf{C}' + \mathbf{a}''\mathbf{s}^\intercal + \mathbf{R}\,, \quad \mathbf{D}'' := \mathbf{D}' + \mathbf{r}\mathbf{b}'^\intercal - \mathbf{R}\,.$$

Alice outputs $\mathbf{a}'', \mathbf{C}''$. Bob outputs $\mathbf{b}'', \mathbf{D}''$.

Lemma A.1. *The protocol in Fig. 22 securely implements $\mathcal{F}_{\mathsf{OLEcor}}$ in Fig. 21 if letting $\ell_1 = k_1 + 4$, $\ell_2 = k_2 + 3$. The security level is $O(2^{-\lambda})$.*

Assuming 2-round OT, the protocol in Fig. 22 takes 4 rounds.

Generation. In the first round, Alice inputs $\mathbf{a}$ to $\mathcal{F}_{\mathsf{OT}}$. In the second round, Bob inputs $M_{\mathbf{b}}, \mathbf{D}_2$ to $\mathcal{F}_{\mathsf{OT}}$, and Alice gets $\mathbf{C}_2$.

In the first round, Bob inputs $\mathbf{b}$ to $\mathcal{F}_{\mathsf{OT}}$. In the second round, Alice inputs $M_{\mathbf{a}}, \mathbf{C}_1$ to $\mathcal{F}_{\mathsf{OT}}$, and Bob gets $\mathbf{D}_1$.

Test. In the third round, Bob sends H. In the last round, Alice sends $\mathbf{v}$.

Extraction and Randomization. In the first round, Alice (resp. Bob) commits $\mathbf{E}, \mathbf{s}$ (resp. $\mathbf{F}, \mathbf{r}, \mathbf{R}$). In the last round, Alice (resp. Bob) opens $\mathbf{E}, \mathbf{s}$ (resp. $\mathbf{F}, \mathbf{r}, \mathbf{R}$).

Moreover, it can be easily optimized to 3-round by the following modification, because $\mathbf{v}$ is a linear function of H if Alice is honest.

Test. In the second round, Bob inputs H to $\mathcal{F}_{\mathsf{OT}}$. In the last round, Alice inputs her strategy to $\mathcal{F}_{\mathsf{OT}}$ so that Bob gets $\mathbf{v}$.

The 3-round version is at least as secure as the 4-round version. From the security perspective, the only difference is that the 3-round version imposes more restriction on corrupted Alice. In the 3-round version, H is hidden from Alice and $\mathbf{v}$ has to be a linear function of H. So it suffices to prove the security with abort of the 4-round version.

Proof of Lemma A.1. The earlier discussion has shown the correctness and the semi-honest security of the protocol. For the malicious security with abort, we start with the simpler case when Bob is corrupted.

Simulation when Bob is corrupted:

- Receive $(\mathbf{b}'', \mathbf{D}'')$ from the ideal functionality.
- In the first round: Get $\mathbf{b}, \mathbf{F}, \mathbf{r}, \mathbf{R}$. Set $\mathbf{b}' = \mathbf{F}\mathbf{b}$. Set $\mathbf{s} = \mathbf{b}'' - \mathbf{b}'$. Set $\mathbf{D}' = \mathbf{D}'' - \mathbf{r}\mathbf{b}'^{\mathsf{T}} + \mathbf{R}$. Sample random $\mathbf{E}$. Sample random $\mathbf{D}_1$ conditioning on $\mathbf{D}' = \mathbf{E}\mathbf{D}_1\mathbf{F}^{\mathsf{T}}$.
- In the second round: Disclose $\mathbf{D}_1$. Get $M^{**}, \mathbf{D}_2$.
- In the third round: Get H. Let $\Delta' = M^{**} - M_{\mathbf{b}}$. Give up if $H\Delta' \cdot (\mathbf{x})_{(2)}, \mathbf{E} \cdot \mathbf{x}$ are not independent for random $\mathbf{x} \leftarrow \mathbb{F}^{\ell_1}$.
- In the last round: Set $\mathbf{v} = -H\Delta' \cdot (\mathbf{a})_{(2)} - H \cdot (\mathbf{D}_1 - \mathbf{D}_2)_{(2)}$ for random $\mathbf{a} \in \mathbb{F}^{\ell_1}$. Disclose $\mathbf{v}, \mathbf{E}, \mathbf{s}$.

In both the simulation and the real execution, $\mathbf{s}, \mathbf{E}, \mathbf{D}_1$ are independent and uniformly random. In the simulation, this is partially because $(\mathbf{b}'', \mathbf{D}'')$ is uniform. In the real execution, $\mathbf{D}_1$ is uniform because it is one-time padded by $\mathbf{C}_1$.

The simulator correctly simulates $\mathbf{v}$ (or equivalently, the leakage $H\Delta' \cdot (\mathbf{a})_{(2)}$) only if $H\Delta' \cdot (\mathbf{x})_{(2)}$ is independent from $\mathbf{E} \cdot \mathbf{x}$ for random $\mathbf{x} \leftarrow \mathbb{F}^{\ell_1}$. Otherwise the simulator gives up.

By Leftover Hash Lemma [HILL99, BDK+11], for random $(\mathbf{E}, \mathbf{x})$, the distribution $(\mathbf{E}, \mathbf{E} \cdot \mathbf{x})$ conditioning on the leakage $H\Delta' \cdot (\mathbf{x})_{(2)}$ is $2^{-(\ell_1 - 2 - k_1)\lambda/2}$-close to uniform. If we set $\ell_1 = k_1 + 4$, the statistical distance is no more than $2^{-\lambda}$. Moreover, since everything is linear over $\mathsf{GF}(2)$, the statistical distance is either 0 or $\geq 1/2$. Thus for most $\mathbf{E}$ (with probability $\geq 1 - 2^{-\lambda+1}$), $\mathbf{E} \cdot \mathbf{x}$ and $\Delta \cdot (\mathbf{x})_{(2)}$ are independent. In other words, the probability that the simulator gives up is no more than $2^{-\lambda+1}$.

Then we analyze the harder case when Alice is corrupted. The corrupted Alice may choose $M^* = M_{\mathbf{a}} + \Delta$ so that Bob gets $\mathbf{D}_1$

$$(\mathbf{D}_1)_{(2)} = M^* \cdot (\mathbf{b})_{(2)} - (\mathbf{D}_1)_{(2)} = (\mathbf{a}\mathbf{b}^\intercal)_{(2)} + \Delta \cdot (\mathbf{b})_{(2)} - (\mathbf{D}_1)_{(2)} \,.$$

Define $\mathbf{L}$ as $(\mathbf{L})_{(2)} = \Delta \cdot (\mathbf{b})_2$, then $\mathbf{C}_1 = \mathbf{a}\mathbf{b}^\intercal + \mathbf{L} - \mathbf{D}_1$.

To pass the test, Alice has to guess $H\Delta \cdot (\mathbf{b})_2$, which equals $H(\mathbf{L})_{(2)}$, correctly. Consider the case $\operatorname{rank}(H\Delta) = \operatorname{rank}\Delta$. In such case, guessing $H(\mathbf{L})_{(2)}$ is equivalent to guessing $\mathbf{L}$. If Alice guesses $\mathbf{L}$ and observes Bob not aborting, she learns $\mathbf{L}$ as a leakage. Since we have

$$\mathbf{a}\mathbf{b}^\intercal = (\mathbf{C}_1 - \mathbf{L}) + \mathbf{D}_1 \,,$$

the simulator can take $\mathbf{C}_1 - \mathbf{L}$ as Alice's "effective $\mathbf{C}_1$".

Simulation when Alice is corrupted:

- Receive $(\mathbf{a}'', \mathbf{C}'')$ from the ideal functionality.
- In the first round: Get $\mathbf{a}, \mathbf{E}, \mathbf{s}$.
- In the second round: Simulate $\mathbf{C}_2$ as uniform. Get $M^*, \mathbf{C}_1$.
- In the third round: Simulates H as uniform. Let $\Delta = M^* - M_{\mathbf{a}}$.

  Do one of the following depending on the rank of Δ and $H\Delta$:

  - If $\operatorname{rank}\Delta \geq \lambda$ and $\operatorname{rank}(H\Delta) \geq \lambda$, send $\perp$ to the ideal functionality[10].
  - If $\operatorname{rank}\Delta \geq \lambda$ and $\operatorname{rank}(H\Delta) < \lambda$, give up.
  - If $\operatorname{rank}\Delta < \lambda$ and $\operatorname{rank}(H\Delta) \neq \operatorname{rank}\Delta$, give up.
  - If $\operatorname{rank}\Delta < \lambda$ and $\operatorname{rank}(H\Delta) = \operatorname{rank}\Delta$, continue.
- In the last round: Sample random $\mathbf{F}$. Give up if $\Delta \cdot (\mathbf{y})_{(2)}, \mathbf{F} \cdot \mathbf{y}$ are not independent for random $\mathbf{y} \leftarrow \mathbb{F}^{\ell_2}$.

  Sample random $\mathbf{b} \leftarrow \mathbb{F}^{\ell_2}$ and set the leakage $\mathbf{L}$ as $(\mathbf{L})_{(2)} = \Delta \cdot (\mathbf{b})_2$.

  Set $\mathbf{r} = \mathbf{a}'' - \mathbf{E} \cdot \mathbf{a}$. Set $\mathbf{R} = \mathbf{C}'' - \mathbf{a}''\mathbf{s}^\intercal - \mathbf{E}(\mathbf{C}_1 - \mathbf{L})\mathbf{F}^\intercal$.

  Disclose $\mathbf{F}, \mathbf{r}, \mathbf{R}$. Receive $\mathbf{v}$. Send $\perp$ to the ideal functionality if

$$\mathbf{v} \neq H \cdot \left((\mathbf{C}_1 - \mathbf{L})_{(2)} - (\mathbf{C}_2)_{(2)}\right) .$$

In both the simulation and the real execution, $\mathbf{C}_2, H, \mathbf{F}, \mathbf{r}, \mathbf{R}$ are independent and uniformly random.

The simulator gives up with probability $O(2^{-\lambda})$. The simulator gives up only if one of the following events happens

- $\operatorname{rank}\Delta \geq \lambda$ and $\operatorname{rank}(H\Delta) < \lambda$,
- $\operatorname{rank}\Delta < \lambda$ and $\operatorname{rank}(H\Delta) \neq \operatorname{rank}\Delta$,
- $\operatorname{rank}\Delta < \lambda$ and $\Delta \cdot (\mathbf{y})_{(2)}, \mathbf{F} \cdot \mathbf{y}$ are not independent for random $\mathbf{y} \leftarrow \mathbb{F}^{\ell_2}$.

[10]The simulation stops after sending $\perp$ to the ideal functionality. A correlated randomness generation protocol has no private input, so there is nothing to hide if the protocol is aborted.

The probability of the first event is bounded by Lemma A.3. The probability of the second event is bounded by Lemma A.2. The probability of the third event is bounded by the Leftover Hash Lemma, as long as $\ell_2 = k_2 + 3$.

If $\mathrm{rank}(H\Delta) \geq \lambda$, Alice will be caught with probability at least $1 - 2^{-\lambda}$ in the real execution. The simulation, in such case, let Bob abort. The corresponding security loss is $O(2^{-\lambda})$.

Finally, consider the case if $\mathrm{rank}(H\Delta) = \mathrm{rank}\,\Delta < \lambda$ and $\Delta \cdot (\mathbf{y})_{(2)}, \mathbf{F} \cdot \mathbf{y}$ are independent for random $\mathbf{y} \leftarrow \mathbb{F}^{\ell_2}$. The leakage is $\mathbf{L}$ simulated correctly, because in the real execution $(\mathbf{L})_{(2)} := \Delta \cdot (\mathbf{b})_2$ is independent from $\mathbf{F} \cdot \mathbf{b}$. □

The proof uses the following two auxiliary lemmas.

Lemma A.2. *For any matrix $\Delta \in \mathsf{GF}(2)^{\ell_1 \times \ell_2}$ such that $\mathrm{rank}\,\Delta \leq \lambda$, sample a random matrix $H \leftarrow \mathsf{GF}(2)^{2\lambda \times \ell_1}$, then*

$$\Pr\Big[\mathrm{rank}(H\Delta) = \mathrm{rank}\,\Delta\Big] \geq 1 - 2^{-\lambda}.$$

Proof. Let $\mathrm{span}(\Delta), \mathrm{span}(H\Delta)$ denote the column space of Δ and $H\Delta$.

The matrix $H\Delta$ has the same rank as Δ if and only if $\mathrm{span}(\Delta)$ is as large as $\mathrm{span}(H\Delta)$. Notice that $\mathbf{y} \mapsto H\mathbf{y}$ is a surjection from $\mathrm{span}(\Delta)$ to $\mathrm{span}(H\Delta)$. The mapping is a bijection if the null space contains only the zero vector. Due to the randomness of H, any non-zero vector falls into the null space with probability $2^{-2\lambda}$. The proof concludes with the union bound.

$$\begin{aligned}
&\Pr\Big[\mathrm{rank}(H\Delta) = \mathrm{rank}\,\Delta\Big] \\
&= \Pr\Big[\mathbf{y} \mapsto H\mathbf{y} \text{ is a bijection from } \mathrm{span}(\Delta) \text{ to } \mathrm{span}(H\Delta)\Big] \\
&= \Pr\Big[\forall \mathbf{v} \in \mathrm{span}(\Delta), \mathbf{v} \neq 0 \implies H\mathbf{v} \neq 0\Big] \\
&\geq 1 - \sum_{\mathbf{v} \in \mathrm{span}(\Delta)} \Pr\Big[\mathbf{v} \neq 0 \text{ and } H\mathbf{v} = 0\Big] \\
&\geq 1 - 2^{\lambda} 2^{-2\lambda} = 1 - 2^{-\lambda}
\end{aligned}$$

□

Lemma A.3. *For any matrix $\Delta \in \mathsf{GF}(2)^{\ell_1 \times \ell_2}$ such that $\mathrm{rank}\,\Delta \geq \lambda$, sample a random matrix $H \leftarrow \mathsf{GF}(2)^{2\lambda \times \ell_1}$, then*

$$\Pr\Big[\mathrm{rank}(H\Delta) \geq \lambda\Big] \geq 1 - 2^{-\lambda}.$$

Proof. It suffices to consider the hardest case when $\mathrm{rank}\,\Delta = \lambda$. The hardest case is proved by Lemma A.2. □