Non-Interactive Secure Computation with Constant Communication Overhead

Yuval Ishai* Ziyang Jin† Naty Peter‡ Akshayaram Srinivasan§

July 29, 2026

Abstract

We study the communication complexity of non-interactive secure computation (NISC) protocols with security against malicious adversaries. We give a general NISC protocol for any two-party function computed by a Boolean circuit C using only $O(|C|\lambda)$ bits of communication, where λ is a computational security parameter. This protocol is unconditionally secure in the random oracle model, assuming a standard random bit OT correlations setup. Compared to Yao's semi-honest protocol, our protocol incurs only a constant communication overhead and achieves security against malicious parties with no additional interaction. Prior works achieved such constant overhead by either using a larger number of rounds or more structured correlations.[1]

1 Introduction

Secure computation [Yao86, GMW87] is a fundamental cryptographic primitive with numerous applications. In the commonly studied scenario involving two parties, one party (the receiver) has a private input x, while the other party (the sender) has a private input y. The receiver aims to obtain the output of a two-party function f applied to (x, y). Secure computation allows the receiver to learn this information without learning anything else about the sender's input, while revealing nothing about the receiver's input to the sender.

In the semi-honest setting, garbled circuits [Yao86] combined with a two-round oblivious transfer (OT) protocol provide a straightforward two-message protocol for this task. In the first round, the receiver encodes its private input x as the receiver's choice bits in the OT protocol. The sender generates the garbled circuit $\widetilde{C}$ along with the set of input labels using the circuit representation of the function. Subsequently, the sender sends $\widetilde{C}$, the labels corresponding to its private input y, and then encodes the labels for the receiver's input wires as the sender's message in the OT protocol. The receiver first decodes the OT message from the sender to obtain the labels corresponding to its input x, and then evaluates the garbled circuit to learn $f(x, y)$. Furthermore, we can replace the two-round OT protocol with random bit-OT correlations, which can be efficiently generated (with

*Technion and AWS. This work is not associated with Amazon.
†University of Toronto.
‡Work done while at the University of Toronto.
§University of Toronto.
[1]This is a full version of the paper in Eurocrypt 2026: https://doi.org/10.1007/978-3-032-25324-8_5

a succinct, reusable setup) using pseudorandom correlation generators [BCG+19] or pseudorandom correlation functions [BCG+20a]. Specifically, after a seed setup phase whose communication cost can be polylogarithmic in the number of OT instances, random OT correlations can be generated on the fly via "silent" local expansion.

The above protocol has a number of appealing features. Firstly, it involves only a single message each from the receiver and the sender. This interaction pattern is minimal: viewing the receiver's message as an encryption of its input, any potential sender only needs to locally compute and send the encrypted output to the receiver. For this reason, the model is sometimes referred to as *non-interactive secure computation* (NISC) [IPS08, IKO+11]. Secondly, this simple semi-honest NISC protocol can be based either on minimal cryptographic assumptions (two-round oblivious transfer) or on a minimal correlated randomness setup of random OT correlations along with a pseudorandom generator (PRG). Thirdly, its communication cost is $O(|C|\lambda)$ bits, where λ is (an exact) computational security parameter and $|C|$ refers to the number of gates in a Boolean circuit C (excluding XOR gates whenever a random oracle is assumed). Unless one relies on "homomorphic" primitives such as fully homomorphic encryption (FHE) [Gen09, BV11, GSW13, BGG+14] or homomorphic secret sharing (HSS) [BGI16, LWYY25, ILL25, MORS25, CHHK25], this is the best communication cost that one could hope for barring a major breakthrough.

Malicious Setting. If the parties are allowed to deviate arbitrarily from the protocol specification, then the state of affairs is not as clean as in the semi-honest setting. For simplicity, we cast all of the following results by assuming an OT correlations setup. Ishai et al. [IPS08, IKO+11] were the first to consider NISC with security against malicious adversaries making a black-box use of cryptography. They achieved a positive result with a multiplicative communication overhead of $\mathrm{polylog}(|C|, \sigma)$ compared to the semi-honest baseline, only making a black-box use of a PRG. (Here and in the following, σ is a statistical security parameter allowing for $2^{-\sigma}$ simulation error.) Subsequent work of Afshar et al. [AMPR14] obtained a concretely efficient version of malicious NISC with multiplicative communication overhead of σ, which is asymptotically worse but concretely better than [IKO+11] for realistic values of σ. Hazay et al. [HIV17] obtained the first malicious constant-round protocols with a constant communication overhead in the random oracle model. However, these protocols either require multiple rounds of interaction or provide a weaker notion of security called security with "correlated abort".[2] Specifically, in this security notion, the sender can force the receiver to abort if the receiver's private input satisfies a sender-defined predicate. (This is similar to, but slightly stronger than, the type of security achieved by constant-round protocols following the "dual execution" framework [HKE12].) Recently, Dittmer et al. [DILO22] presented malicious NISC protocols with constant communication overhead, but relied on more structured "OLE-style" correlations that are stronger than OT correlations and are much more expensive to generate [BCG+19, BCG+20b]. See Table 1 for a summary of prior work.

The above state of the art leaves the following natural question open:

Is there a malicious-secure NISC protocol with communication cost $O(|C|\lambda)$ assuming random OT correlations and black-box use of PRG?

Such a protocol would bridge the asymptotic gap between what is known in a semi-honest setting and what is known in a malicious setting.

[2] A constant-overhead NISC protocol with correlated abort was implicitly given in [IKO+11] as a building block towards a fully secure protocol with a higher overhead. The correlated-abort protocol from [HIV17] offers better concrete efficiency and uses a sublinear number of string-OTs.

Work	Assumptions	Communication Cost	Security
[IKO+11]	OT correlations + PRG	$O(\|C\|\lambda \cdot \text{polylog}(\|C\|, \sigma))$	Standard
[AMPR14]	2-round batch-committing OT	$O(\|C\| \cdot \lambda\sigma)$	Standard
[IKO+11, HIV17]	OT correlations + RO	$O(\|C\|\lambda)$	Correlated-Abort
[DILO22]	programmable OLE + RO	$O(\|C\|\lambda)$	Standard
Ours	OT correlations + RO	$O(\|C\|\lambda)$	Standard

Table 1: Comparison of NISC protocols in prior works, where λ is an exact computational security parameter, and σ is a statistical security parameter. OT correlations mean random bit-OT correlations.

Our Result. For the question above, we show that if we replace a PRG by a random oracle, we can indeed close the above gap.

Theorem 1.1. *Assume random bit-OT correlations and a random oracle. There is a (malicious-secure, two-party) NISC protocol for any Boolean circuit C with a communication cost of $O(|C|\lambda)$ bits, where λ is an exact computational security parameter.*

1.1 Related Work

Following the pioneering works of Mohassel and Franklin [MF06] and Lindell and Pinkas [LP07] on "black-box" malicious-secure variants of Yao's protocol, there has been a long line of work [HKE12, NO16, WRK17, HIV17, KRRW18, HSS17, DILO22, CWYY23, GLOS25] on constructing constant-round, malicious-secure two-party protocols with constant communication overhead when compared to semi-honest Yao. The most recent work along this line is due to Cui et al. [CWYY23], who gave a 10-round protocol in the VOLE-hybrid plus random oracle model that has $2\times$ overhead when compared to semi-honest Yao.

An orthogonal line of work, originating from the "row reduction" garbled circuit optimization of Naor et al. [NPS99], focused on improving the concrete size of garbled circuits in the random oracle model [KS08, ZRE15, HK20, RR21]. For standard Boolean circuits, the state of the art is a garbled circuit construction whose size is $1.5\lambda \cdot N_{\mathsf{AND}}$ where N_{AND} denotes the number of nonlinear 2-input gates in the circuit (e.g., AND) [RR21]. This has been shown to be nearly optimal with recent lower bounds that apply to natural restricted classes of "gate-by-gate" garbled circuit constructions [JRR25, XHX25].

Chase et al. [CDI+19] introduced a notion called reusable non-interactive secure computation. In such a protocol, the first-round message sent by the receiver could be reused an unbounded number of times with (potentially) different senders. They constructed a reusable NISC protocol in the OLE-hybrid model. The work of Ishai et al. [IKSS23] constructed a reusable NISC assuming a 2-round semi-honest OT plus a random oracle.

Ishai et al. [IKSS22a, IKSS22b, IKSS23] extended the standard NISC protocol to a two-sided NISC protocol. In the setting of two-sided NISC, both the sender and the receiver speak in each round of the interaction, and at the end of two rounds, they both receive the output $f(x, y)$. In the malicious setting, simply running two instances of one-sided NISC in opposite directions does not work, as nothing prevents the parties from using inconsistent inputs in the two executions.

2 Technical Overview

In this section, we give a high-level overview of the techniques used in constructing a NISC protocol with constant communication overhead.

Bird's-eye view of our Techniques. Starting with the early works of Mohassel-Franklin [MF06] and Lindell-Pinkas [LP07], there has been a long line of work on minimizing the communication overhead of black-box constant-round secure computation. Here, we focus on NISC protocols with constant overhead, and achieve this using OT correlations (a minimal setup) and a random oracle. Previous works with the same setup [IKO$^+$11, HIV17, IKSS22a] employed a similar set of techniques, including the IPS watchlist mechanism, but all faced technical barriers that prevented them from achieving the current goal. The first two could only achieve constant overhead with selective failures (satisfying relaxed security with "correlated abort"), and the latter required super-constant overhead due to the incompatibility of packing techniques with their technical approach. The key innovation in our work lies in a new way of using packed secret sharing along with sublinear proof systems to circumvent these previous barriers. The most novel technical aspect is the development of a black-box method to combine Ligero proofs with the watchlist mechanism. We explain these in detail below.

Starting point. Our NISC protocol builds on the IPS compiler [IPS08]. The IPS compiler gives a generic way to construct constant-round, two-party protocols with security against malicious adversaries by combining three building blocks: an outer protocol, an inner protocol, and a watchlist.

The outer protocol is a client-server secure computation protocol between two clients (namely, the sender and the receiver) and m servers. The clients send an encoding of their inputs to the servers, the servers then perform local computation, and finally they send an encoding of the output to the receiver client. The receiver client decodes the server messages to learn the output of the function. The outer protocol has security against any malicious adversary that corrupts one of the clients and a constant fraction of the servers.

In the IPS compiler, the servers in the outer protocol are virtually emulated by the clients using a semi-honest secure inner protocol. To be a bit more precise, each server in the outer protocol is emulated using an instance of the inner protocol. The private inputs used by the receiver and the sender in the i-th instance of the inner protocol correspond to the messages they send to the i-th server in the outer protocol. At the end of this emulation, the receiver client learns the messages that the servers intend to send to it in the outer protocol. This guarantees correctness. However, the inner protocol is only guaranteed to be secure against semi-honest adversaries, whereas a malicious sender or receiver could cheat in all the inner protocol executions, thereby violating the security of the outer protocol. This could allow the corrupted client to learn the private input of the honest client. This kind of cheating is prevented by the watchlist.

A watchlist protocol prevents a corrupted client from cheating in a large number of inner protocol instances. Specifically, in the watchlist protocol, each client chooses a secret subset of the inner protocol executions as its "watchlist". The watchlist protocol delivers to a client the input and the randomness of the other client in each execution that is present in its watchlist. This allows this client to check consistency between the messages that it received in the inner protocol with the input and randomness it got via the watchlist protocol. If any inconsistency is detected, the honest client aborts. Since the set of executions in the watchlist is chosen uniformly at random,

if no cheating is detected in the watchlist executions, it can be argued that a large fraction of the inner protocol executions are honestly emulated. Thus, we can rely on the security of the outer protocol to argue that a corrupted client learns no information about the inputs of the honest client except the output of the function.

The IPS compiler gives a constant-round protocol when one uses a constant-round outer protocol, a constant-round inner protocol, and a constant-round watchlist protocol. Ishai et al. [IKSS21, IKSS22a] gave a round-optimal approach to instantiate the IPS compiler. Specifically, they used a two-round client-server protocol from the work of [IKP10] as the outer protocol, and the two-round semi-honest secure protocol from garbled circuits [Yao86] as the inner protocol. Their main contribution is to weaken the requirement from the watchlist protocol. Specifically, they showed that the set of watchlist executions need not be secret and they can be revealed to the corrupted client after it sends its messages in the inner protocol. This allowed them to instantiate the watchlist in the random oracle model using the Fiat-Shamir heuristic.

While the above works achieved round optimality, their communication cost was far from being optimal. The communication complexity in these works was $\mathsf{poly}(|C|, \lambda)$, a far cry from that of the semi-honest protocol that achieved $O(|C|\lambda)$ bits. Our goal in this work is to bridge this gap. We achieve this via a novel combination of a specialized outer protocol and an inner protocol that is tailored to emulate the server computations in the outer protocol with minimal overhead. We again use the Fiat-Shamir approach to instantiate the watchlist.

2.1 Our Outer Protocol

The outer protocol in our work computes an authenticated version of the BMR garbled circuits [BMR90]. Specifically, it gets an additive share of the wire mask from the sender and the receiver, a MAC key from the receiver, the wire labels from the sender and the precomputed PRG values on the wire labels from the sender and generates an authenticated version of the BMR garbled gate. We make a couple of observations on the garbled gate computation:

- Given the precomputed PRG values from the sender, each garbled gate entry can be computed via a constant-degree polynomial with a constant number of monomials.
- The MAC prevents a cheating sender from sending incorrect PRG computations. If the sender cheats, then the MAC check will fail.
- Having the wire mask secret shared between the sender and the receiver prevents the sender from mounting an input-dependent abort attack against the receiver.

From these three observations, we conclude that it is sufficient for the outer protocol to compute a constant-degree polynomial on the sender and receiver inputs.

Semi-honest protocol. Let's for now assume that this outer protocol needs to only be semi-honest secure. In this case, the sender and the receiver can compute Shamir shares of their private inputs and send these shares to the servers. Let d be the degree of the function to be computed and let t be the corruption threshold for the outer protocol. Assume that the number of servers is $O(dt)$. The servers can locally compute a degree-d function on the Shamir shares to get a dt-sharing of the output. They can then send these shares to the receiver[3] who reconstructs the garbled gate entries from the server messages.

[3]For the sake of keeping the exposition simple, we are ignoring the random shares of 0 that need to be added to the dt-sharing to prevent the receiver from gleaning more information about the sender inputs.

Consider the communication cost of this protocol. Fix the finite field size to be 2^λ. The number of private inputs for the sender and the receiver to compute the BMR garbled gate entries are $O(|C|)$ field elements. The threshold t needs to be fixed as $O(\lambda)$. Thus, the total number of servers needed is $O(\lambda)$ as d is a constant. Therefore, the sender and the receiver communicate $O(|C|\lambda)$ field elements in total to the servers. Since each field element is $O(\lambda)$ bits, the total communication between the clients and the servers is $O(|C|\lambda^2)$ bits. As these messages are going to be the inputs to the inner protocol, this serves as a lower bound on the communication cost of our compiler. Therefore, when compared to semi-honest Yao, the outer protocol already incurs an $O(\lambda)$ multiplicative overhead.

Packing to the rescue. To reduce the communication cost of the outer protocol, we rely on packing techniques introduced by Franklin and Yung [FY92]. Recall how packed Shamir secret sharing works. Let $\mathbb{F}$ denote the underlying field. We set $|\mathbb{F}| \geq 2^\lambda$. Fix a set of ℓ distinct elements $\beta_1, \ldots, \beta_\ell \in \mathbb{F}$ and fix another disjoint set of m distinct elements $\alpha_1, \ldots, \alpha_m \in \mathbb{F}$. Let $\overline{s} := (s_1, \ldots, s_\ell) \in \mathbb{F}^\ell$ be a vector of secrets to share. To hide the secrets, we sample a random univariate polynomial $p(\cdot)$ of degree $t + \ell$ conditioned on $p(\beta_i) = s_i$ for all $i \in [\ell]$. To compute the shares, for every $j \in [m]$, party j's share is computed as $x_j := p(\alpha_j)$. The vector consisting of all shares $\overline{x} := (x_1, \ldots, x_m) \in \mathbb{F}^m$ is often called a *sharing* of $\overline{s}$. Sometimes we write $[\overline{s}]$ to denote a sharing of $\overline{s}$. When we add/multiply the shares corresponding to a packed secret sharing scheme, we obtain entry-wise addition/product of the secret vectors under the hood. We now explain how to compute constant-degree polynomials using packed secret sharing.

Case of Degree 2. Suppose the outer protocol was tasked to compute only a degree-2 function on the receiver's and the sender's private inputs. Let vector $\overline{x}$ and vector $\overline{y}$ denote the private inputs of the receiver and the sender, respectively. The outer protocol is constructed as follows:

- *Step 1.* Let $\odot$ denote the entry-wise product operation. Let p be a degree-2 polynomial p, and we represent it by a vector of its monomials. Here we abuse the notation p to also mean a vector. We decompose the monomials in p as $p^x \odot p^y$ where vector p^x (resp. p^y) only contains variables in $\overline{x}$ (resp. $\overline{y}$) and constants. For example, for polynomial $p(\overline{x}, \overline{y}) := 1+y_1+x_1y_3+x_2^2+x_3y_2+y_3y_4$, and $\overline{x} := (x_1, x_2, x_3)$ and $\overline{y} := (y_1, y_2, y_3, y_4)$, the decomposition is

$$\begin{aligned} p &= \begin{pmatrix} 1 & y_1 & x_1y_3 & x_2^2 & x_3y_2 & y_3y_4 \end{pmatrix}, \\ p^x &= \begin{pmatrix} 1 & 1 & x_1 & x_2^2 & x_3 & 1 \end{pmatrix}, \\ p^y &= \begin{pmatrix} 1 & y_1 & y_3 & 1 & y_2 & y_3y_4 \end{pmatrix}. \end{aligned}$$

- *Step 2.* The receiver and the sender send a packed sharing of p^x and p^y respectively to the servers. The servers locally multiply the shares to obtain shares of the monomials in the polynomial p. To mask the monomials, the sender additionally sends a sharing of a random size-$|p|$ vector $\overline{z}$ conditioned on the sum of its elements being zero. The servers add the sharing of $\overline{z}$ to the sharing of p and send the result to the receiver. In the end, the receiver reconstructs the monomials of $p + \overline{z}$ and adds them together to obtain evaluation of p.

Reducing constant degree to degree 2. Once we have an outer protocol for degree-2 polynomials, we can reduce the computation of any constant-degree polynomial to a degree-2 polynomial

as we only have two clients. Specifically, for any constant degree polynomial, the clients can decompose the terms in each monomial from their respective private inputs. For instance, if a monomial is $x_2y_1y_3$, then the client decomposes this to $p^x = x_2$ and $p^y = y_1y_3$. They construct the vector p^x and p^y as above and send packed secret sharing of p^x and p^y to the servers. The rest of the protocol follows identically to the case of degree 2.

Communication Cost. Suppose we used the protocol based on packed secret sharing to compute BMR garbled gate entries with a packing factor of $\ell = O(\lambda)$. The receiver and the sender in this case need to send $O(|C|/\lambda)$ shares to the servers where each share packs $O(\lambda)$ secrets. The number of servers is $O(\lambda)$, and it takes λ bits to represent a field element. Therefore, the total communication cost is $O(|C|/\lambda) \cdot \lambda \cdot O(\lambda) = O(|C|\lambda)$ bits.

Extending to malicious adversaries. Note that the above outer protocol is only semi-honest secure but we need it to be secure against malicious adversaries for the IPS compiler. We observe that a malicious client can cheat in the following two ways in the above protocol:

- **Sending incorrectly generated packed shares.** The malicious client could cheat by sending shares that do not lie on a degree-$(t + \ell)$ polynomial. In the case of [IKP10], this was dealt by using a special secret sharing scheme called pairwise verifiable secret sharing and a conditional disclosure of secrets protocol that prevents the adversary from learning any information about the honest party's inputs if it sent inconsistent shares to the honest servers. Unfortunately, this approach incurs at least an $O(\lambda)$ multiplicative overhead in the individual share size and this is something that we cannot afford.
- **Sending shares of incorrectly decomposed values.** The malicious client could send packed shares of p^x or p^y that are incorrectly decomposed. For instance, in the places where there needs to be a constant 1, it can send an arbitrary value. In places where the term needs to x_1x_2, it can send another value and so on. We need to ensure that clients are sending packed shares of correctly decomposed values.

We explain how to deal with these two issues below.

Issue 1: Incorrectly generated packed shares. Observe that a packed Shamir secret sharing can be viewed as a Reed-Solomon codeword, where each server holds a symbol of the codeword. A server receiving an inconsistent symbol from the malicious client can be viewed as a corrupted position in the received codeword. If we can ensure that the number of corrupted positions is within the error-correction distance, then the correct computation result can still be recovered from the result of honest servers' computation. Thus, it comes down to the problem of testing Reed-Solomon proximity—given a received codeword, accept if the received codeword disagrees with its closest exact Reed-Solomon codeword on at most t positions, and reject otherwise. There has been a lot of research on testing Reed-Solomon proximity [PS94, BS05]. We use the Ligero proof system [AHIV17] that tests Reed-Solomon proximity of multiple codewords together. However, we cannot use the Ligero proof system directly due to a technical issue that we explain next.

Note that each client shares multiple vectors. Let n be the number of vectors and we view them as a matrix of n rows where each row contains the purported codeword sent by a corrupted client. We need to test Reed-Solomon proximity of n codewords together, which seemingly can be achieved by a direct application of Ligero. However, the notion of distance used in Ligero is the maximum distance between any row of the purported codeword to an actual Reed-Solomon code.

This distance notion turns out to be insufficient for our purpose. To see why this is the case, let's denote this distance to be d. Consider a purported codeword sent by the corrupted client such that the first d positions of the first row is inconsistent with a Reed-Solomon codeword, the second d positions of the second row is inconsistent with a Reed-Solomon codeword, and so on. In total, there are dn positions across all the rows that are inconsistent with a Reed-Solomon codeword. In the IPS compiler, the servers that correspond to these inconsistent positions are corrupted. This means that the number of servers needs to be larger than dn, which is something we cannot afford.

What we really need is the union over the positions across each row that is different from an actual codeword to have small cardinality. If this was the case, we can then corrupt those servers in the outer protocol in the IPS compiler. Thus, in our paper, we define the *distance* between n received codewords and their closest exact RS codewords to be the cardinality of the union of corrupted positions over all n codewords. Under this new definition of distance, we use the result of [BCI+20] to prove the soundness of Ligero with improved parameters. This allows us to argue that all the corruptions caused by the malicious client is concentrated on a small subset of servers and therefore error-correction can still be achieved.

At a high level, the Ligero proof for RS proximity views the sharing of all n vectors as an $n \times m$ matrix U as each RS codeword has length m. Each server $j \in [m]$ obtains a column of U. The matrix U is committed column-wise by a straight-line extractable commitment scheme, which can be constructed in the random oracle model [Pas03]. The Ligero proof essentially sends two things: a random linear combination of the rows of U, and openings for a random subset of columns. The intuition is that a random linear combination of RS codewords is a RS codeword, whereas a random linear combination of non-codewords is unlikely to be a codeword. The openings for a random subset of columns allows the verifier to check if the random linear combination is computed honestly.

For communication cost, the resulting codeword of the random linear combination takes $m \cdot \lambda$ bits given our finite field $\mathbb{F}$ of order 2^λ, and openings of columns takes $O(m) \cdot n \cdot \lambda$ bits. The number of vectors n in the outer protocol is $O(|C|/\ell)$. Therefore, the communication cost is $O(m|C|\lambda/\ell)$. Taking packing size $\ell = O(\lambda)$ and the number of servers $m = O(\lambda)$, we obtain the communication complexity $O(|C|\lambda)$ for the Ligero proof.

Issue 2: Incorrectly decomposed values. To compute the BMR garbled gate entries, we need to compute a degree-3 polynomial. This means that decomposed vectors p^x or p^y can have degree at least 2 or higher, which makes it non-trivial to test.

To overcome this, we further decompose p^x into vectors $p^x_1, \ldots, p^x_d$ and decompose p^y into vectors $p^y_1, \ldots, p^y_d$ such that each of p^x_i (resp. p^y_i) is linear. As a result, each element of p^x_i (resp. p^y_i) is either a variable from x (resp. y) or a constant. These vectors should satisfy the decomposition requirement $p^x = p^x_1 \odot \cdots \odot p^x_d$ and $p^y = p^y_1 \odot \cdots \odot p^y_d$, which can be checked by the watchlist.

Now that p^x_i (resp. p^y_i) is linear, we can enforce a set of linear constraints on these vectors to ensure that the corresponding slots are aligned correctly for entry-wise product. For example, suppose we have the following decomposition for p^x:

$$
\begin{aligned}
p^x &= \begin{pmatrix} 1 & x_1x_2x_3 & x_2^2x_4 & x_2x_4 \end{pmatrix}, \\
p^x_1 &= \begin{pmatrix} 1 & x_1 & x_2 & x_2 \end{pmatrix}, \\
p^x_2 &= \begin{pmatrix} 1 & x_2 & x_2 & x_4 \end{pmatrix}, \\
p^x_3 &= \begin{pmatrix} 1 & x_3 & x_4 & 1 \end{pmatrix}.
\end{aligned}
$$

For each p_i^x, let $p_i^x[j]$ denote the j-th element of the vector. Then, we need to ensure that $p_1^x[3] = p_1^x[4] = p_2^x[2] = p_2^x[3]$ since all these positions hold the same variable x_2, and $p_2^x[4] = p_3^x[3]$ since both positions hold x_4, and $p_1^x[1] = p_2^x[1] = p_3^x[1] = p_3^x[4] = 1$ since these positions should contain the constant 1. These equalities can be translated to a set of linear equations. After this high-degree to linear reduction, we can directly apply the Ligero proof for linear constraints to ensure that the decomposed values p^x and p^y are generated honestly.

Another issue that arises with the BMR garbling process and our outer protocol is that the inputs and the masking bits are boolean values, but our outer protocol performs arithmetic operations over a finite field $\mathbb{F}$. A malicious client may use vectors whose positions are supposed to contain boolean values to actually contain some arbitrary values in $\mathbb{F}$. To deal with this, consider the equation $x(1-x) = 0$ over $\mathbb{F}$, which enforces the value of x to be 0 or 1. Following this idea, for a vector v, we compute $v_{\mathsf{bool}} := v \odot (1 - v)$ where 1 denotes the all-one vector.[4] For the positions in v that need to have value 0 or 1, the corresponding positions in v_{bool} must be 0. Note that other positions in v_{bool} are irrelevant and can be ignored. This again gives us a set of linear equations $v_{\mathsf{bool}}[j] = 0$ for all j where $v[j]$ is supposed to be a boolean value. Again, we use the Ligero proof for linear constraints to ensure the boolean values are converted honestly to 0/1 in $\mathbb{F}$.

The communication cost of Ligero proof for linear constraints is also $O(|C|\lambda)$. The difference from the Ligero proof for RS proximity is that instead of sending a linear combination of codewords, the Ligero proof for linear constraints sends a random linear combination of polynomials by sending the coefficients of the resulting polynomial. The resulting polynomial will have degree asymptotically the same as m (the codeword length). Therefore, the communication cost is still dominated by the size of opening of a random subset of columns. Hence, the communication cost is still $O(|C|\lambda)$.

2.2 Our Inner Protocol

The semi-honest inner protocol performs server computation in the outer protocol. In the outer protocol, the receiver (resp. sender) client sends packed secret shares of p_1^x, p_2^x, p_3^x (resp. p_1^y, p_2^y, p_3^y) to the servers as computing a boolean circuit can be reduced to computing degree-3 polynomials. Denote the sharings of these vectors by $[p_1^x], [p_2^x], [p_3^x]$ (resp. $[p_1^y], [p_2^y], [p_3^y]$). Additionally, the sender sends packed secret shares of $\mathbf{z}$, which is a vector of additive secret sharing of zeros to mask the monomials. Denote the sharing of $\mathbf{z}$ by $[\mathbf{z}]$. The server function does two things:

1. It computes $[p^x] = [p_1^x] \odot [p_2^x] \odot [p_3^x]$ and $[p^y] = [p_1^y] \odot [p_2^y] \odot [p_3^y]$.
2. It computes $[p] = [p^x] \odot [p^y] + [\mathbf{z}]$.

For the first operation, observe that $[p^x]$ (resp. $[p^y]$) can be locally computed by the receiver (resp. sender). Thus, the receiver (resp. sender) client also sends a precomputed $[p^x]$ (resp. $[p^y]$) to the servers. We will use the watchlist to ensure that the pre-computation is performed honestly.

For the second operation, suppose the sharings $[p^x], [p^y], [\mathbf{z}]$ are vectors of k elements, the evaluation $[p^x] \odot [p^y] + [\mathbf{z}]$ can be viewed as k parallel oblivious linear evaluations (OLEs).

We note that if we were given k preprocessed random OLE correlations over $\mathbb{F}$ where $|\mathbb{F}| = 2^\lambda$, this can be done trivially. However, generating random OLE correlations is much more expensive

[4] Note that we can enforce the correct computation of v_{bool} in a similar way to enforcing the correct computation of p^x and p^y.

than for instance, generating random bit-OT correlations. We would like our inner protocol to build on basic assumptions like standard random bit-OT correlations.

To solve this problem, we make use of *multiplication-friendly embeddings* (MFEs) [CCXY18, EXY22]. Essentially, an MFE is a pair of linear maps (ρ, μ) where ρ maps a vector of elements in a small ring to a single element in a big ring, and μ maps an element in a big ring to a vector of elements in a small ring. They satisfy $x \cdot y = \rho(\mu(x) \odot \mu(y))$ where x and y are elements from the big ring. We use the result of [CRX21] which states that there exists an explicit MFE family

$$\left(\rho_\lambda : \mathbb{F}_2^{r(\lambda)} \to \mathbb{F}_{2^\lambda},\ \mu_\lambda : \mathbb{F}_{2^\lambda} \to \mathbb{F}_2^{r(\lambda)}\right)_{\lambda \in \mathbb{N}}$$

with ratio $r(\lambda)/\lambda \to 5.12$ as $\lambda \to \infty$. This allows us to express an OLE in $\mathbb{F}_{2^\lambda}$ by $O(\lambda)$ OTs in $\mathbb{F}_2$. Using the linearity property of MFEs, we construct an inner protocol that computes k OLEs in $\mathbb{F}_{2^\lambda}$ using $O(k\lambda)$ number of preprocessed random bit-OT correlations. The communication cost of our protocol is only a constant multiple of the trivial protocol based on OLE correlations. We remark that similar protocols have appeared in [BGMN18].

For communication cost, the vectors p^x, p^y contain $O(|C|)$ field elements and are shared via a packed Shamir secret sharing of packing size ℓ. Thus, we have $k = O(|C|/\ell)$. Each field element takes λ bits to represent. Therefore, the inner protocol has communication cost $O(|C|\lambda/\ell)$. The outer protocol has m servers, and each server runs an instance of the inner protocol. Hence, the total communication cost of the inner protocols used in the outer protocol is $O(|C|\lambda)$ when we take $m = O(\lambda)$ and $\ell = O(\lambda)$.

3 Preliminaries

Mathematical notation. For $n \in \mathbb{N}$, we use $[n]$ to denote the set of elements $\{1, 2, \ldots, n\}$. We use $x \leftarrow S$ to mean that x is drawn uniformly at random from the set S. We use $\odot$ to denote the entry-wise product (also called Hadamard product or Schur product) operation of two vectors. For any matrix U, we use U_i to denote the i-th row of U, and we use $U[j]$ to denote the j-th column of U. For a string x, we use $|x|$ to denote the number of bits in x. For a vector v over a finite field $\mathbb{F}$, we use $|v|$ to denote the number of elements in v. All the logs in this paper are by default base 2.

Cryptography notation. Let λ denote the cryptographic security parameter, and we assume that all cryptographic algorithms implicitly take 1^λ as an input parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$, there exists a λ_0 such that for all $\lambda \geq \lambda_0$, we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function. We use the abbreviation PPT to mean probabilistic polynomial time. We say that two distribution ensembles $\{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $\{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if for every non-uniform probabilistic polynomial time (PPT) distinguisher $\mathcal{D}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|\Pr[\mathcal{D}(1^\lambda, X_\lambda) = 1] - \Pr[\mathcal{D}(1^\lambda, Y_\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$. We use $\{X_\lambda\}_{\lambda \in \mathbb{N}} \approx_c \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ to denote the two ensembles are computationally indistinguishable.

3.1 Secure Two-Party Computation

We use standard definitions of standalone secure two-party computation from Goldreich's book [Gol04]. The definitions below largely follow Chapter 7 of [Gol04]. We give definitions for both semi-honest security and malicious security.

Definition 3.1 (Semi-Honest Security)**.** *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, and $f_1(x,y)$ (resp. $(f_2(x,y))$) denote the first (resp. second) element of $f(x,y)$. Let π be a two-party protocol for computing f. The view of the first (resp. second) party during an execution of π on (x,y), denoted view_1^π (resp. view_2^π), is $(x, r, m_1, \ldots, m_t)$ (resp. $(y, r, m_1, \ldots, m_t)$), where r represents the first (resp. second) party's internal randomness, and m_i represents the i-th message it receives. Let $\mathsf{output}^\pi = (\mathsf{output}_1^\pi, \mathsf{output}_2^\pi)$ denote the output of π.*

We say that protocol π securely computes f in the presence of semi-honest adversaries if there exist probabilistic polynomial-time algorithms $\mathcal{S}_1, \mathcal{S}_2$ such that

$$\{(\mathcal{S}_1(x, f_1(x,y)),\ f(x,y))\}_{x,y} \approx_c \{(\mathsf{view}_1^\pi(x,y),\ \mathsf{output}^\pi(x,y))\}_{x,y},$$

and

$$\{(\mathcal{S}_2(y, f_2(x,y)),\ f(x,y))\}_{x,y} \approx_c \{(\mathsf{view}_2^\pi(x,y),\ \mathsf{output}^\pi(x,y))\}_{x,y},$$

where $x, y \in \{0,1\}^*$ and $|x| = |y|$.

Definition 3.2 (Malicious Security)**.** *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, and $f_1(x,y)$ (resp. $(f_2(x,y))$) denote the first (resp. second) element of $f(x,y)$. Let Π be a two-party protocol for computing f.*

Let $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ be a pair of probabilistic polynomial-time algorithms representing strategies in the real model. Such a pair is admissible with respect to Π (for the real malicious model) if at least one of $\mathcal{A}_1, \mathcal{A}_2$ coincides with the strategy specified by Π. (In particular, this A_i ignores the auxiliary input z). The joint execution of Π under $\overline{\mathcal{A}}$ in the real model (on input pair (x,y) and auxiliary input z), denoted $\mathsf{REAL}_{\Pi,\overline{\mathcal{A}}(z)}(x,y)$, is defined as the output pair resulting from the interaction between $\mathcal{A}_1(x,z)$ and $\mathcal{A}_2(y,z)$.

Let $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$ be a pair of probabilistic polynomial-time algorithms representing strategies in the ideal model. Such a pair is admissible (in the ideal malicious model) if at least one $i \in \{1,2\}$ is honest. The joint execution of f under $\overline{\mathcal{S}}$ in the ideal model (on input pair (x,y) and auxiliary input z), denoted $\mathsf{IDEAL}_{f,\overline{\mathcal{S}}(z)}(x,y)$, is defined by uniformly selecting a random tape r for the adversary, and letting $\mathsf{IDEAL}_{f,\overline{\mathcal{S}}(z)}(x,y) = \Gamma(x,y,z,r)$, where $\Gamma(x,y,z,r)$ is defined as follows. When party 1 is honest, define $\Gamma(x,y,z,r) := (f_1(x,y'), \mathcal{S}_2(y,z,r,f_2(x,y')))$ where y' is chosen by the adversary based on y,z,r. Additionally, if the adversary corrupting party 2 sends an abort signal to f, then $\Gamma(x,y,z,r) := (\perp, \mathcal{S}_2(y,z,r,f_2(x,y'),\perp))$. When party 2 is honest, $\Gamma(x,y,z,r) := (\mathcal{S}_1(x,z,r,f_1(x',y)), f_2(x',y))$ where x' is chosen by the adversary based on x,z,r.

We say that Π securely computes f in the presence of malicious adversaries if for every probabilistic polynomial-time pair of algorithms $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ that is admissible for the real model, there exists a probabilistic polynomial-time pair of algorithms $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$ that is admissible for the ideal model such that

$$\{\mathsf{IDEAL}_{f,\overline{\mathcal{S}}(z)}(x,y)\}_{x,y,z} \approx_c \{\mathsf{REAL}_{\Pi,\overline{\mathcal{A}}(z)}(x,y)\}_{x,y,z},$$

where $x, y, z \in \{0,1\}^*$ such that $|x| = |y|$ and $|z| = \mathsf{poly}(|x|)$.

In our paper, we consider the case where only party 1 outputs. We call party 1 the receiver, denoted by R, and call party 2 the sender, denoted by S. We focus on two-round protocols, and we denote a two-party, two-round protocol by a tuple of algorithms $(\Pi_1, \Pi_2, \mathsf{out}_\Pi)$. Π_1 is run by the receiver and takes input x and outputs (π_1, sk), where π_1 is the first-round message and sk is the secret key. Once the sender receives π_1, it runs $\Pi_2(\pi_1, y)$ and outputs π_2 to the receiver. The receiver runs $\mathsf{output}_\Pi(\pi_2, sk)$ and obtains the output.

Our paper also considers the random oracle model. In the random oracle model, both parties have access to a shared random oracle $\mathcal{H} : \mathcal{D} \to \mathcal{R}$, and each party is allowed to make polynomially many queries in the domain $\mathcal{D}$ to the random oracle during the execution of the protocol. For a new query, $\mathcal{H}$ samples a uniform random element from its range $\mathcal{R}$. An element in $\mathcal{R}$ has length $\mathsf{poly}(\lambda)$. For a query that has already been asked, $\mathcal{H}$ returns the previous answer to this query. In the definition of semi-honest security, the view_i^π now additionally includes the queries made by party i to the random oracle and the corresponding answers, and the protocol becomes $\pi^{\mathcal{H}}$ and the PPT algorithms $\mathcal{S}_1, \mathcal{S}_2$ become $\mathcal{S}_1^{\mathcal{H}}, \mathcal{S}_2^{\mathcal{H}}$ who also have access to the random oracle. In the definition of malicious security, in the real execution, the protocol Π now has access to the random oracle and becomes $\Pi^{\mathcal{H}}$, and the adversary becomes $\mathcal{A}_i^{\mathcal{H}}$; in the ideal execution, the PPT algorithm now becomes $\mathcal{S}_i^{\mathcal{H}}$. Additionally, in both semi-honest and malicious security definitions, the PPT simulator $\mathcal{S}_i^{\mathcal{H}}$ can program the random oracle to return a desired answer to a specific query before the query has been asked. If the query has already been asked and the simulator tries to program the random oracle on that query, the simulator will abort.

3.2 Oblivious Transfer (OT) and OT Correlation

In an Oblivious Transfer (OT) protocol, there is a sender and a receiver. The sender holds two bits $x_0, x_1 \in \{0,1\}$ and the receiver holds a selection bit $b \in \{0,1\}$. The ideal functionality $\mathcal{F}_{\mathsf{OT}}$ is defined as follows: The sender sending x_0, x_1 to $\mathcal{F}_{\mathsf{OT}}$, and the receiver sends b to $\mathcal{F}_{\mathsf{OT}}$. Then $\mathcal{F}_{\mathsf{OT}}$ sends x_b to the receiver.

We consider a two-round OT protocol Π_{OT} that implements $\mathcal{F}_{\mathsf{OT}}$. In the first round, the receiver sends a message to the sender. In the second round, the sender responds. As a result, the receiver obtains x_b without learning about x_{1-b}, and the sender does not learn b.

In a random bit-OT correlation, the ideal functionality $\mathcal{F}_{\mathsf{ROT}}^N$ is defined as follows: For every $j \in [N]$, it samples $r_{j,0}, r_{j,1}, b_j \in \{0,1\}$ uniformly at random. Then it sends the receiver $(b_j, r_{j,b})$ and it sends the sender $(r_{j,0}, r_{j,1})$. As a result, the receiver obtains $\{(b_j, r_{j,b})\}_{j\in[N]}$ and the sender obtains $\{(r_{j,0}, r_{j,1})\}_{j\in[N]}$. Sometimes we make N implicit and just write $\mathcal{F}_{\mathsf{ROT}}$ for the random bit-OT correlations ideal functionality.

We extend the definitions of secure two-party computation above to the $\mathcal{F}_{\mathsf{ROT}}$-hybrid model. In this model, in the real execution, in the setup phase of the protocol, the receiver and the sender receive random bit-OT correlations from $\mathcal{F}_{\mathsf{ROT}}$. When we consider the simulator for the corrupted party, the simulator receives exactly the setup information that a corrupted party would receive. The simulator does not receive the honest party's setup output. Note that for random OT correlations, the simulator can sample the honest party's setup values: If the receiver is corrupted, the simulator receives (b, r_b) from $\mathcal{F}_{\mathsf{ROT}}$, then the simulator samples $r_{1-b} \leftarrow \{0,1\}$ and sets (r_b, r_{1-b}) as the honest sender's setup. If the sender is corrupted, the simulator receives (r_0, r_1) from $\mathcal{F}_{\mathsf{ROT}}$, then the simulator samples $b \leftarrow \{0,1\}$ and sets (b, r_b) as the honest receiver's setup.

In the semi-honest security definition, the view now includes the party's input, the party's received setup values from $\mathcal{F}_{\mathsf{ROT}}$, the messages the party receives, and the queries the party makes to the random oracle and the corresponding answers. In the malicious security definition, the adversary may choose its input after seeing the OT correlation values, or it can uses its OT correlations inconsistently during the execution of the protocol. However, the adversary cannot learn or change the OT correlation values of the honest party.

3.3 Client-Server Protocol

We consider a client-server protocol [DI06] with two clients and m servers. Furthermore, only the receiver client outputs. We assume the standard model of secure point-to-point channels between each client and the servers. The servers do not communicate with each other. While this is an instance of general secure computation, with special interaction and restricted functionalities, it will be convenient to use the following explicit syntax.

Let $f(\overline{x}, \overline{y})$ be an arbitrary two-party functionality, where the receiver has input vector $\overline{x}$ and the sender has input vector $\overline{y}$. A two-client, m-server, two-round MPC protocol Φ for computing f can be described by a tuple of algorithms $((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$, where

- $\mathsf{Share}_1(1^\lambda, \overline{x})$: It takes the receiver's input $\overline{x}$, and outputs shares $(\hat{x}_1, \ldots, \hat{x}_m)$ and a verification key vk. Note that the share $\hat{x}_j$ will be sent to server $j \in [m]$ and the verification key vk will be kept privately to the receiver.
- $\mathsf{Share}_2(1^\lambda, \overline{y})$: It takes the sender's input $\overline{y}$, and outputs shares $(\hat{y}_1, \ldots, \hat{y}_m)$, where the share $\hat{y}_j$ is sent to server $j \in [m]$.
- $\mathsf{Eval}(1^\lambda, j, (\hat{x}_j, \hat{y}_j))$: It takes as input the shares $\hat{x}_j, \hat{y}_j$ sent to server j where $j \in [m]$, and outputs a string ϕ_j to the receiver client. We call Eval the server evaluation function.
- $\mathsf{Dec}(1^\lambda, vk, (\phi_1, \ldots, \phi_m))$: It takes ϕ_j from all servers $j \in [m]$ and the verification key vk, and outputs either a string $\overline{z} = f(\overline{x}, \overline{y})$ or a special symbol $\perp$ to indicate abort. Sometimes, the verification key vk is optional if it is not used in the algorithm.

In our case, each server performs the same server evaluation function. Specifically, the two-round protocol Φ runs as follows:

1. In the first round of the protocol, the receiver runs $\mathsf{Share}_1(1^\lambda, \overline{x})$ to obtain the first round messages $(\hat{x}_1, \ldots, \hat{x}_m)$ and a verification key vk. At the same time, the sender runs $\mathsf{Share}_2(1^\lambda, \overline{y})$ to obtain its first round messages $(\hat{y}_1, \ldots, \hat{y}_m)$. For each $j \in [m]$, the receiver (resp. sender) sends $\hat{x}_j$ (resp. $\hat{y}_j$) to server j.

2. In the second round, each server j where $j \in [m]$ runs the server evaluation function $\mathsf{Eval}(1^\lambda, j, \hat{x}_j, \hat{y}_j)$ to obtain a string ϕ_j, and then sends ϕ_j to the receiver client.

3. Upon receiving ϕ_j for all $j \in [m]$, the receiver runs $\mathsf{Dec}(1^\lambda, vk, (\phi_1, \ldots, \phi_m))$, which reconstructs the result from the shares $\phi_1, \ldots, \phi_m$ and then checks the validity using the verification key vk. If the check fails, the receiver outputs $\perp$. Otherwise, the receiver may perform some further computation on the result and eventually outputs $\overline{z} = f(\overline{x}, \overline{y})$.

3.4 Oblivious Linear Evaluation (OLE)

Let $\mathbb{F}$ be a finite field. In an Oblivious Linear Evaluation (OLE) protocol, there is a sender and a receiver. The sender holds $a, b \in \mathbb{F}$ that defines an affine function $f(x) = ax + b$, and the receiver holds input $x \in \mathbb{F}$. The receiver first sends a message to the sender and then the sender responds. As a result, the receiver learns function evaluation $f(x)$ while the sender does not learn x and the receiver does not learn anything about the function $f(\cdot)$ beyond what can be inferred from $f(x)$.

3.5 Extractable Commitments in ROM

Following [IKSS22a], our protocol compilers use non-interactive, straight-line extractable commitments in the random oracle model. Namely, the commitments are computationally hiding and

straight-line extractable by observing the queries that the adversary makes to the random oracle. Such commitments were constructed in [Pas03].

3.6 Packed Shamir Secret Sharing

Packed secret sharing was introduced by Franklin and Yung [FY92]. Let $\mathbb{F}$ be a finite field and secrets and polynomials below are all over $\mathbb{F}$. A $(t, t+\ell+1; \ell, m)$-packed secret sharing scheme shares a vector of ℓ secrets over m parties. It satisfies that no subset of at most t parties can deduce anything about the value of the secrets in an information-theoretic sense, and any subset of at least $t+\ell+1$ parties can reconstruct all the secrets. We use $+$ and $\cdot$ to denote the addition and multiplication operations over the finite field $\mathbb{F}$.

We briefly explain how packed Shamir secret sharing [FY92] is constructed. Fix a set of ℓ distinct elements $\beta_1, \ldots, \beta_\ell \in \mathbb{F}$ and fix another disjoint set of m distinct elements $\alpha_1, \ldots, \alpha_m \in \mathbb{F}$. Note that this requires the finite field $\mathbb{F}$ to have at least $m+\ell$ distinct elements. Let $s := (s_1, \ldots, s_\ell) \in \mathbb{F}^\ell$ be a vector of secrets to share. To hide the secrets, sample a random univariate polynomial p of degree $k = (t+\ell)$ conditioned on $p(\beta_i) = s_i$ for all $i \in [\ell]$. To compute the shares, for every $j \in [m]$, party j's share, denoted by x_j, is computed as $x_j := p(\alpha_j)$. We call the vector of all shares $x := (x_1, \ldots, x_m) \in \mathbb{F}^m$ a *sharing* of s. A sharing of s is also commonly denoted by $[s]$.

We also note that packed Shamir secret sharing has a robust reconstruction scheme. Suppose at most t shares are corrupted, as long as $t \leq \lfloor (m-k-1)/2 \rfloor$, a unique secret can still be recovered using the Berlekamp-Welch [WB86] algorithm.

The packed Shamir secret sharing is a *linear* secret sharing scheme. Let x be a packed Shamir secret sharing of vector s and let x' be a packed Shamir secret sharing of another vector s'. It is easy to observe that $ax + x'$ is a packed Shamir secret sharing of $as + s'$ for any $a \in \mathbb{F}$. Additionally, the entry-wise product of two sharings $x \odot x'$ gives a sharing of $s \odot s'$ over a polynomial of higher degree.

We can also view a packed Shamir secret sharing as a variant of Reed-Solomon code. Let $\overline{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$ and let $\overline{\beta} = (\beta_1, \ldots, \beta_\ell) \in \mathbb{F}^\ell$. We define the variant of Reed-Solomon code $\mathsf{RS}_{\overline{\alpha},\overline{\beta}}[\mathbb{F}, k, n]$ as follows. For any message $m := (m_1, \ldots, m_\ell) \in \mathbb{F}^\ell$, we sample a random degree-k polynomial p conditioned on $p(\beta_i) = m_i$ for all $i \in [\ell]$, and output the codeword $(p(\alpha_1), \ldots, p(\alpha_n))$.

3.7 Non-Interactive Zero-knowledge Arguments (NIZK)

A non-interactive zero-knowledge argument system is a pair of probabilistic polynomial-time algorithms (P, V). Let $L \in \mathsf{NP}$ be a language and let $\mathcal{R}_L$ be the NP relation for L. To prove $(x, w) \in \mathcal{R}_L$, the prover P on input (x, w) outputs a proof π to the verifier. Then the verifier V on input (x, π) outputs accept or reject. The argument system (P, V) must satisfy three properties.

- *Completeness.* For any $(x, w) \in \mathcal{R}_L$, the verifier accepts the honest prover's proof π with probability 1.
- *Soundness.* If $x \notin L$, for any PPT cheating prover P^*, the verifier V accepts with negligible probability in $|x|$.
- *Zero-knowledge.* Since the argument system is non-interactive, we only need honest-verifier zero-knowledge. For any $(x, w) \in \mathcal{R}_L$, there exists a PPT simulator S such that S on input x outputs π^*, and π^* is computationally indistinguishable from π, the proof produced by an honest prover P in the real execution.

Sometimes, the argument system $(P, V)^{\mathcal{H}}$ additionally assumes a random oracle $\mathcal{H}$, where both the prover and the verifier can query the random oracle polynomial number of times.

4 Outer Protocol

In this section, we progressively define three client-server protocols [DI05]. The first protocol computes an array of degree-2 polynomials (i.e. quadratics); the second protocol computes an array of constant-degree polynomials; the third protocol computes any two-party functionality f modeled as a Boolean circuit C. The security of these three protocols is defined against specific adversaries that will be described later, which we call "admissible adversaries". Looking ahead, we will use the watchlist method from the IPS compiler [IPS08] and a set of non-interactive zero-knowledge arguments based on Ligero [AHIV17] to lift the security from admissible adversaries to standard malicious adversaries.

Conforming Protocol. Our outer protocol is set in the client-server model with two clients and m servers. A two-client, m-server, two-round protocol Φ for computing f can be described by a tuple of algorithms $((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$. Additionally, our protocol is *conforming*, meaning that the first-round messages sent from the clients to the servers are shares of a packed Shamir secret sharing scheme. Formally,

Definition 4.1 (Conforming Protocol)**.** *A two-round, two-client, m-server protocol Φ is said to be* $\mathsf{conforming}$ *if the output of algorithm Share_1 (resp. Share_2) is of the form $\{(x_{i,1}, \ldots, x_{i,m})\}_{i\in[n]}$ (resp. $\{(y_{i,1}, \ldots, y_{i,m})\}_{i\in[n]}$) where n is the number of vectors to share, and for each $i \in [n]$, $(x_{i,1}, \ldots, x_{i,m})$ (resp. $(y_{i,1}, \ldots, y_{i,m})$) are the shares of a packed Shamir secret sharing scheme.*

4.1 Protocol for Computing Quadratics

Notations. Fix a finite field $\mathbb{F}$ with order 2^λ. Denote the receiver's input by $\overline{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ and the sender's input by $\overline{y} = (y_1, \ldots, y_n) \in \mathbb{F}^n$. Let $\overline{p} = (p_1, \ldots, p_N)$ where $N \in \mathbb{N}$ be a vector of degree-2 multivariate polynomials over $\mathbb{F}$ to be computed by the protocol. For $i \in [N]$, each polynomial $p_i(\overline{x}_i, \overline{y}_i)$ takes as input vector $\overline{x}_i \subseteq \overline{x}$ and vector $\overline{y}_i \subseteq \overline{y}$. For each $i \in [N]$, we use $|p_i|$ to denote the number of monomials in p_i. Each polynomial p_i is represented in form of a sum of its monomials, denoted by $p_i = m_{i,1} + \ldots + m_{i,|p_i|}$, where the monomials $m_{i,j}$ for $j \in \{1, \ldots, |p_i|\}$ are ordered from low-degree to high-degree and in lexicographic order. We abuse the notation p_i to also mean a vector of its monomials, i.e., $p_i := (m_{i,1}, \ldots, m_{i,|p_i|})$. Furthermore, we abuse the notation $\overline{p} := (m_{1,1}, \ldots, m_{1,|p_1|}, \ldots, m_{N,1}, \ldots, m_{N,|p_N|})$ to mean a long vector of all the monomials of all polynomials concatenated together and let $N_{\mathsf{mono}} = |p_1| + \ldots + |p_N|$ be the total number of monomials in $\overline{p}$. Let $(\mathsf{Share}_{(t,\ell,m)}, \mathsf{Rec}_{(t,\ell,m)})$ be a $(t, t+\ell+1; \ell, m)$-packed Shamir secret sharing scheme over $\mathbb{F}$, where ℓ is the packing size. This means that any t parties gain no information about the secret, while any $t+\ell+1$ parties can recover the secret. For any positive natural number $a, b \in \mathbb{N}$, let $(\mathsf{Share}_{(at+b\ell,\ell,m)}, \mathsf{Rec}_{(at+b\ell,\ell,m)})$ denote a $(at + b\ell, at + b\ell + \ell + 1; \ell, m)$-packed Shamir secret sharing scheme.

Decomposition. We describe a deterministic procedure, denoted by $\mathsf{Decomp}(\overline{p})$, that decomposes the polynomials $\overline{p} = (p_1, \ldots, p_N)$ into entry-wise products of linear polynomials. The decomposition works as follows:

Step 1. Recall that $\odot$ is the entry-wise product operation. For each polynomial p_i where $i \in [N]$, we first decompose it as $p_i = p_i^x \odot p_i^y$ where p_i^x (resp. p_i^y) only contains variables in $\overline{x}_i$ (resp. $\overline{y}_i$) or constants. For example, let $p_i(\overline{x}_i, \overline{y}_i) = 1 + y_1 + x_1 y_3 + x_2^2 + x_5 y_2 + y_3 y_5$ for some i, and $\overline{x}_i = (x_1, x_2, x_5)$ and $\overline{y}_i = (y_1, y_2, y_3, y_5)$, then the decomposition is

$$\begin{aligned} p_i &= \begin{pmatrix} 1 & y_1 & x_1 y_3 & x_2^2 & x_5 y_2 & y_3 y_5 \end{pmatrix}, \\ p_i^x &= \begin{pmatrix} 1 & 1 & x_1 & x_2^2 & x_5 & 1 \end{pmatrix}, \\ p_i^y &= \begin{pmatrix} 1 & y_1 & y_3 & 1 & y_2 & y_3 y_5 \end{pmatrix}. \end{aligned}$$

Step 2. We further decompose each of p_i^x and p_i^y into $p_i^x = p_{i,1}^x \odot p_{i,2}^x$ and $p_i^y = p_{i,1}^y \odot p_{i,2}^y$ respectively. For each $k \in \{1, 2\}$, each $p_{i,k}^x$ (resp. $p_{i,k}^y$) contains elements of either a single variable in $\overline{x}_i$ (resp. $\overline{y}_i$) or a field constant. To continue the example above, the further decomposition will be

$$\begin{aligned} p_{i,1}^x &= \begin{pmatrix} 1 & 1 & x_1 & x_2 & x_5 & 1 \end{pmatrix}, & p_{i,1}^y &= \begin{pmatrix} 1 & y_1 & y_3 & 1 & y_2 & y_3 \end{pmatrix}, \\ p_{i,2}^x &= \begin{pmatrix} 1 & 1 & 1 & x_2 & 1 & 1 \end{pmatrix}, & p_{i,2}^y &= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & y_5 \end{pmatrix}. \end{aligned}$$

To make the decomposition unique, one can specify a fixed set of decomposition rules. Therefore, each degree-2 polynomial p_i is eventually decomposed into 6 vectors $p_i^x, p_{i,1}^x, p_{i,2}^x, p_i^y, p_{i,1}^y, p_{i,2}^y$. We concatenate the decomposed vectors of all polynomials p_i^x (resp. p_i^y) for $i \in [N]$ into 6 long vectors $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x, \overline{p}^y, \overline{p}_1^y, \overline{p}_2^y$ as follows:

$$\begin{aligned} \overline{p}^x &= p_1^x || \ldots || p_N^x, & \overline{p}_1^x &= p_{1,1}^x || \ldots || p_{N,1}^x, & \overline{p}_2^x &= p_{1,2}^x || \ldots || p_{N,2}^x, \\ \overline{p}^y &= p_1^y || \ldots || p_N^y, & \overline{p}_1^y &= p_{1,1}^y || \ldots || p_{N,1}^y, & \overline{p}_2^y &= p_{1,2}^y || \ldots || p_{N,2}^y. \end{aligned}$$

Observe that $\overline{p}^x = \overline{p}_1^x \odot \overline{p}_2^x$ and $\overline{p}^y = \overline{p}_1^y \odot \overline{p}_2^y$. Hence, the final decomposition can be expressed as $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x, \overline{p}^y, \overline{p}_1^y, \overline{p}_2^y \leftarrow \mathsf{Decomp}(\overline{p})$.

Linear Constraints from the Decomposition. To ensure that each client in the MPC protocol honestly executes the decomposition described above, we can enforce a set of linear constraints on $\overline{p}_1^x, \overline{p}_2^x, \overline{p}_1^y, \overline{p}_2^y$. Let us use a minimal example to illustrate. Suppose the protocol computes two degree-2 polynomials $\overline{p} = (p_1, p_2)$ where $p_1 = x_1 y_1$ and $p_2 = x_2^2 + x_1 y_2 + y_1^2$. Then the vector representation $\overline{p} = (x_1 y_1, x_2^2, x_1 y_2, y_1^2)$ and the decomposition is

$$\begin{aligned} \overline{p}^x &= \begin{pmatrix} x_1 & x_2^2 & x_1 & 1 \end{pmatrix}, & \overline{p}^y &= \begin{pmatrix} y_1 & 1 & y_2 & y_1^2 \end{pmatrix}, \\ \overline{p}_1^x &= \begin{pmatrix} x_1 & x_2 & x_1 & 1 \end{pmatrix}, & \overline{p}_1^y &= \begin{pmatrix} y_1 & 1 & y_2 & y_1 \end{pmatrix}, \\ \overline{p}_2^x &= \begin{pmatrix} 1 & x_2 & 1 & 1 \end{pmatrix}, & \overline{p}_2^y &= \begin{pmatrix} 1 & 1 & 1 & y_1 \end{pmatrix}. \end{aligned}$$

We describe the linear constraints that the receiver should satisfy below, and the constraints for the sender can be derived similarly. To express the linear constraints, let

$$v := (\overline{p}_1^x || \overline{p}_2^x) = (x_1, x_2, x_1, 1, 1, x_2, 1, 1)$$

be a (column) vector of 8 elements. Let $v[i]$ denote the i-th element of v. Observe a correct decomposition should satisfy the following constraints:

- The variables need to be correctly aligned, which means the same variable should take the same value across the decomposed vectors. For the example above, for variable x_1 it should satisfy $v[1] = v[3]$, and for variable x_2 it should satisfy $v[2] = v[6]$.

- The "1" slots should take value 1. For the example above, it should satisfy $v[4] = v[5] = v[7] = v[8] = 1$.

The above constraints can be converted into a set of linear equations. We can also write the constraints in matrix form $Av = b$ where A is a $|v| \times |v|$ matrix and b is a vector of size $|v|$.

For the sender, the linear constraints are similar. Additionally, the sender also needs to use additive secret sharings of zero to mask the resulting monomials such that they will not be exposed to the receiver. Specifically, for each p_i where $i \in [N]$, the sender samples a vector $p_i^0 = (r_{i,1}, \ldots, r_{i,|p_i|})$ of random field elements over $\mathbb{F}$ conditioned on the elements of p_i^0 summing to zero. Let $|p_i|$ denote the number of elements in p_i^0. This introduces an additional linear constraint $\sum_{j=1}^{|p_i|} r_{i,j} = 0$. Let $\overline{p}^0 = p_1^0 || \ldots || p_N^0$ denote the concatenation of all p_i^0's. We formally define *linear constraints* over a set of n vectors $v_1, \ldots, v_n$ as follows:

Definition 4.2 (Linear Constraints)**.** *Let $\mathbb{F}$ be a finite field and let ℓ, n be natural numbers. Consider n vectors $v_1, \ldots, v_n \in \mathbb{F}^\ell$. The* linear constraints *$\mathcal{R}$ on the set of vectors $\{v_1, \ldots, v_n\}$ are a system of linear equations $Av = b$, where $v = (v_1 || \ldots || v_n) \in \mathbb{F}^{n\ell}$ is the concatenation of all vectors, and $A \in \mathbb{F}^{n\ell \times n\ell}$ is a sparse matrix and $b \in \mathbb{F}^{n\ell}$.*

Let $\mathcal{R}_x$ denote the linear constraints on the receiver's vectors $\overline{p}_1^x, \overline{p}_2^x$. Similarly, let $\mathcal{R}_y$ denote the linear constraints on the sender's vectors $\overline{p}_1^y, \overline{p}_2^y, \overline{p}^0$, where $\overline{p}^0$ is a vector of additive secret sharing of zeros. Let $\mathcal{R} := (\mathcal{R}_x, \mathcal{R}_y)$ be the combined linear constraints. Note that the linear constraints ensure that the adversary must use consistent values for its variables. For example, if variable x_1 appears in both p_1 and p_2, then the receiver must plug in the same value for x_1 in these two polynomials. On the other hand, if $\overline{p}_1^x, \overline{p}_2^x$ satisfy the linear constraints $\mathcal{R}_x$, then a unique input vector $\overline{x}$ can be extracted from the receiver's decomposed vectors $\overline{p}_1^x, \overline{p}_2^x$. Similarly, if $\mathcal{R}_y$ is satisfied, then a unique input vector $\overline{y}$ can be extracted from the sender's decomposed vectors $\overline{p}_1^y, \overline{p}_2^y$.

Admissible Adversary. Let m be the number of servers in the client-server MPC protocol. Let $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x$ (resp. $\overline{p}^y$, $\overline{p}_1^y$, $\overline{p}_2^y$, $\overline{p}^0$) be the vectors to be packed secret shared by the receiver (resp. sender) as a result of the decomposition described above. For each $j \in [m]$, let $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}\}$ (resp. $\{\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{0}_j\}$) be the first-round messages sent by the receiver (resp. sender) to server j, which are the purported shares for $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x$ (resp. $\overline{p}^y, \overline{p}_1^y, \overline{p}_2^y, \overline{p}^0$).

Definition 4.3 (Admissible Adversary)**.** *An adversary $\mathcal{A}$ corrupting either client and a set of servers T ($|T| \leq t$) is said to be admissible w.r.t. linear constraints $\mathcal{R} = (\mathcal{R}_x, \mathcal{R}_y)$ if*

1. *(Consistency Constraints) The first-round messages $\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}$ sent by the receiver (resp. $\hat{y}_j$, $\hat{y}_{j,1}$, $\hat{y}_{j,2}, \hat{0}_j$ sent by sender) to the honest servers $j \in [m] \setminus T$ are consistent with packed Shamir secret sharings of vectors $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x$ (resp. $\overline{p}^y, \overline{p}_1^y, \overline{p}_2^y, \overline{p}^0$).*

2. *(Decomposability Constraints) The first-round messages $\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}$ sent by the receiver (resp. $\hat{y}_j$, $\hat{y}_{j,1}$, $\hat{y}_{j,2}$ sent by sender) satisfy $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2}$ (resp. $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2}$) for all honest servers $j \in [m] \setminus T$.*

3. *(Linear Constraints) Assuming the consistency constraint is satisfied, let $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x$ (resp. $\overline{p}^y, \overline{p}_1^y, \overline{p}_2^y, \overline{p}^0$) be the secret vectors extracted from the first-round messages sent by the receiver (resp. sender) to the honest servers. Then, the vectors $\overline{p}_1^x, \overline{p}_2^x$ (resp. $\overline{p}_1^y, \overline{p}_2^y, \overline{p}^0$) satisfy the linear constraints $\mathcal{R}_x$ (resp. $\mathcal{R}_y$).*

Description of the Protocol. Consider a two-client, m-server conforming MPC protocol (see Definition 4.1) where one client and at most t servers can be corrupted. Before entering the protocol, both parties execute the decomposition procedure $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x, \overline{p}^y, \overline{p}_1^y, \overline{p}_2^y \leftarrow \mathsf{Decomp}(\overline{p})$. Note that $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x, \overline{p}^y, \overline{p}_1^y, \overline{p}_2^y$ contain entries that consist of variables (symbols) and constants, which are known to both parties. However, the vectors $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x$ (resp. $\overline{p}^y, \overline{p}_1^y, \overline{p}_2^y$) become private once the receiver (resp. sender) assigns the values from its private input $\overline{x}$ (resp. $\overline{y}$) into these vectors.

The conforming protocol $\Phi_{\mathsf{quad}}(\overline{p}, \overline{x}, \overline{y}) := ((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$ for computing degree-2 polynomials $\overline{p} = (p_1, \ldots, p_N)$ is described in Fig. 1 below. We note that if N_{mono} does not divide ℓ, we pad the vector $\overline{p}$ such that N is a multiple of ℓ.

Recall that there are m servers, and at most t servers can be corrupted. Since the final resulting vectors are recovered using a $(4t + 3\ell, 4t + 3\ell + \ell + 1; \ell, m)$-packed Shamir secret sharing scheme, we need to use a polynomial of degree $4t + 4\ell$ to share the secrets. If we view the sharing as a Reed-Solomon code, then the distance of the code is $m - (4t + 4\ell)$. In order to correct t errors, we need $t \leq \lfloor (m - (4t + 4\ell) - 1)/2 \rfloor \implies m \geq 6t + 4\ell + 1$.

Lemma 4.4. *Let ℓ be the packing size of packed secret sharing scheme and let number of servers $m \geq 6t + 4\ell + 1$. The two-client, m-server, two-round conforming protocol Φ_{quad} for computing a vector of degree-2 polynomials is secure against an admissible adversary $\mathcal{A}$ corrupting either client and up to t servers.*

Communication Complexity. Recall that in the protocol, N' is the number of size-ℓ short vectors, and $N_{\mathsf{mono}} = N' \cdot \ell$ is the total number of monomials over all polynomials $p_1, \ldots, p_N$ to evaluate. By packed secret sharing, the receiver sends $3N'$ field elements and the sender sends $4N'$ field elements to each server. Each server evaluates and then sends N' elements back to the receiver. Therefore, the total number of field elements communicated is $8N'm$. Plugging in $N' = N_{\mathsf{mono}}/\ell$, we have $8N'm = 8mN_{\mathsf{mono}}/\ell$. Each field element takes λ bits. Thus, the communication complexity is $O(mN_{\mathsf{mono}}\lambda/\ell)$. If each polynomial in $p_1, \ldots, p_N$ contains constant number of monomials (which means $N_{\mathsf{mono}} = O(N)$), and the number of servers $m = O(\ell)$, then the communication complexity is $O(N\lambda)$.

Security Proof. We formally define the security of Φ_{quad} against the admissible adversary as follows.

Definition 4.5 (Security Against Admissible Adversary)**.** *Let f be a two-party, single-output functionality. A conforming two-round, two-client, m-server protocol $\Phi = ((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$ for computing f is secure against admissible adversaries corrupting up to t servers if for any admissible adversary $\mathcal{A}$ (see Definition 4.3) w.r.t. linear constraints $\mathcal{R}$ corrupting either client and at most t servers, there exists an ideal world simulator Sim_Φ such that for any choice of inputs of the honest client, the following two distributions are computationally indistinguishable:*

- ***Real Execution.*** *The admissible adversary $\mathcal{A}$ interacts with the honest parties that follow the protocol specification. The output of the real execution consists of the output of the admissible adversary $\mathcal{A}$ and the output of the honest client. If the sender is the honest client, then the sender does not output.*
- ***Ideal Execution.*** *This corresponds to the ideal world interaction where Sim_Φ and the honest client have access to a trusted party implementing f. The honest client sends its input to f. The honest output client outputs whatever the trusted functionality sends back. If the receiver*

Protocol Φ_{quad} for Computing Quadratics:

- **Inputs:** Receiver has input $\overline{x}$ and sender has input $\overline{y}$. The common input $\overline{p} = (p_1, \ldots, p_N)$ is a vector of degree-2 polynomials to evaluate. Let N_{mono} be the total number of monomials in $p_1, \ldots, p_N$. If N_{mono} is not a multiple of ℓ, we pad $\overline{p}$ with zeros to make N_{mono} divide ℓ.
- **Setup:** Both parties execute $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2, \overline{p}^y, \overline{p}^y_1, \overline{p}^y_2 \leftarrow \mathsf{Decomp}(\overline{p})$ described above.
- **Round 1:**

  The receiver runs $\mathsf{Share}_1(1^\lambda, \overline{x})$ as follows:

  1. It assigns values of monomials in $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2$ using $\overline{x}$.
  2. It partitions each of $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2$ into $N' := N_{\mathsf{mono}}/\ell$ short vectors of size ℓ. Denote these vectors by $\overline{v}^x_1, \ldots, \overline{v}^x_{N'}$ and $\overline{v}^x_{1,1}, \ldots, \overline{v}^x_{N',1}$ and $\overline{v}^x_{1,2}, \ldots, \overline{v}^x_{N',2}$ respectively.
  3. For each $i \in [N']$,
     (a) It computes $[\overline{v}^x_{i,1}] = \mathsf{Share}_{(t,\ell,m)}(\overline{v}^x_{i,1})$ and $[\overline{v}^x_{i,2}] = \mathsf{Share}_{(t,\ell,m)}(\overline{v}^x_{i,2})$, where $[\overline{v}^x_{i,1}] := (s_{i,1,1}, \ldots, s_{i,m,1})$ denotes a sharing of $\overline{v}^x_{i,1}$, and $[\overline{v}^x_{i,2}] := (s_{i,1,2}, \ldots, s_{i,m,2})$ denotes a sharing of $\overline{v}^x_{i,2}$.
     (b) It computes $[\overline{v}^x_i] = [\overline{v}^x_{i,1}] \odot [\overline{v}^x_{i,2}]$, where $[\overline{v}^x_i] := (s_{i,1}, \ldots, s_{i,m})$ denotes a sharing of $\overline{v}^x_i$.

     For $j \in [m]$, let $\hat{x}_{j,1} := (s_{1,j,1}, \ldots, s_{N',j,1})$, $\hat{x}_{j,2} := (s_{1,j,2}, \ldots, s_{N',j,2})$, $\hat{x}_j := (s_{1,j}, \ldots, s_{N',j})$.
  4. It outputs $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}\}_{j\in[m]}$.

  The sender runs $\mathsf{Share}_2(1^\lambda, \overline{y})$ as follows:

  1. It assigns values of monomials in $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2$ using $\overline{y}$.
  2. It samples N vectors $p^0_1, \ldots, p^0_N$ such that for each $i \in [N]$, vector $p^0_i := (r_{i,1}, \ldots, r_{i,|p_i|})$ contains $|p_i|$ random field elements under the constraint $\sum_{j=1}^{|p_i|} r_{i,j} = 0$ (i.e. an additive secret sharing of 0). Let $\overline{p}^0 := p^0_1 || \ldots || p^0_N$.
  3. It partitions each of $\overline{p}^0, \overline{p}^y, \overline{p}^y_1, \overline{p}^y_2$ into $N' := N_{\mathsf{mono}}/\ell$ short vectors of size ℓ. Denote these vectors by $\overline{v}^0_1, \ldots, \overline{v}^0_{N'}$ and $\overline{v}^y_1, \ldots, \overline{v}^y_{N'}$ and $\overline{v}^y_{1,1}, \ldots, \overline{v}^y_{N',1}$ and $\overline{v}^y_{1,2}, \ldots, \overline{v}^y_{N',2}$ respectively.
  4. For each $i \in [N']$, it computes the sharings $[\overline{v}^y_{i,1}], [\overline{v}^y_{i,2}], [\overline{v}^y_i]$ similarly as above and $[\overline{v}^0_i] = \mathsf{Share}_{(4t+3\ell,\ell,m)}(\overline{v}^0_i)$ where $[\overline{v}^0_i]$ denotes a sharing of $\overline{v}^0_i$. For $j \in [m]$, define $\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{0}_j$ similarly as above.
  5. It outputs $\{\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{0}_j\}_{j\in[m]}$.

- **Round 2:** Server j runs $\mathsf{Eval}(1^\lambda, j, (\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}), (\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{0}_j))$ for $j \in [m]$ as follows:

  1. **Verify:** It verifies that $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2}$ and $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2}$. Note that this check will eventually be replaced by the watchlist.
  2. **Eval:** It outputs $\phi_j := \hat{x}_j \odot \hat{y}_j + \hat{0}_j$ via N' parallel OLE operations, where ϕ_j is a vector of N' field elements.

- **Output:** The receiver runs $\mathsf{Dec}(1^\lambda, (\phi_1, \ldots, \phi_m))$ as follows:

  1. It reconstructs $\overline{o} = \mathsf{Rec}_{(4t+3\ell,\ell,m)}(\phi_1, \ldots, \phi_m)$, where $\overline{o} := \overline{o}_1 || \ldots || \overline{o}_{N'}$.
  2. Note that $\overline{o}$ is a vector of $N_{\mathsf{mono}} = N' \cdot \ell$ monomials. It re-partitions the monomials in $\overline{o}$ back into N vectors of size $|p_1|, \ldots, |p_N|$ in order. Denote the re-partitions by $\overline{o} = \overline{f}_1 || \ldots || \overline{f}_N$, where $\overline{f}_i := (f_{i,1}, \ldots, f_{i,|p_i|})$, which corresponds to evaluations of monomials in p_i for $i \in [N]$.
  3. For each $i \in [N]$, it computes $z_i := \sum_{j=1}^{|p_i|} f_{i,j}$.
  4. It outputs $\overline{z} := (z_1, \ldots, z_N)$ as the evaluations of polynomials.

Figure 1: Protocol for Computing Quadratics

is the honest client, Sim_Φ *sends a special instruction to the trusted functionality to either give the output of* f *to the receiver or the special symbol* $\perp$*. If the sender is the honest client, then it has no output. The output of the ideal execution corresponds to the output of* Sim_Φ *and the output of the honest client.*

Proof of Lemma 4.4. Let $\mathcal{A}$ be an admissible adversary that corrupts either client and up to t servers. Let $\mathsf{Sim}_\Phi = (\mathsf{Sim}^r_\Phi, \mathsf{Sim}^s_\Phi)$ be the simulator in the ideal world, where Sim^r_Φ (resp. Sim^s_Φ) is the simulator when $\mathcal{A}$ corrupts the receiver (resp. sender) client and up to t servers. Let $F(\overline{x}, \overline{y})$ denote the ideal functionality that on input $\overline{x}, \overline{y}$, outputs the evaluations of degree-2 polynomials $p_1, \ldots, p_N$. Let $T \subseteq [m]$ denote the set of corrupted servers where $|T| \leq t$.

The receiver is corrupted. The simulator $\underline{\mathsf{Sim}^r_\Phi}$ works as follows:

1. It sends (on behalf of the honest sender) default random values as purported secret shares to each corrupted server in T.

2. (Extract Input) It receives the first-round messages sent to the honest servers from $\mathcal{A}$. Note that the shares sent by the admissible adversary $\mathcal{A}$ to the honest servers are consistent with packed Shamir secret sharings. Using the shares sent to the honest servers, it recovers unique secret vectors $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2$ since the number of servers m is big enough such that t is less than the error-correction distance. Since the vectors $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2$ shared by the admissible adversary satisfy the linear constraints $\mathcal{R}_x$, the input $\overline{x}$ used by $\mathcal{A}$ is uniquely defined. Thus, Sim^r_Φ extracts the corrupted receiver's purported input $\overline{x}$ from the recovered secret vectors.

3. (Craft Output) It sends $\overline{x}$ to the ideal functionality F and obtains the output $\overline{f} = (f_1, \ldots, f_N)$, which are the evaluations of the polynomials in $\overline{p} = (p_1, \ldots, p_N)$. Using these evaluations, Sim^r_Φ samples the vector $\overline{o}' = \overline{f}'_1 || \ldots || \overline{f}'_N$ as follows: for each $i \in [N]$, it samples $\overline{f}'_i = (f_{i,1}, \ldots, f_{i,|p_i|})$ that contains $|p_i|$ random field elements under the constraint $\sum_{j=1}^{|p_i|} f_{i,j} = f_i$. Next, Sim^r_Φ re-partitions $\overline{o}'$ as $\overline{o}' = \overline{o}'_1 || \ldots || \overline{o}'_{N'}$, where each vector $\overline{o}'_i$ for $i \in [N']$ has length ℓ.

4. (Simulate Corrupted Servers) By calling $\mathsf{Extrapolate}$ on the shares sent by $\mathcal{A}$ to the honest servers, it obtains the shares $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}\}_{j \in T}$ for the corrupted servers that are consistent with the shares sent to the honest servers.

   Then, Sim^r_Φ simulates the corrupted servers' computation using the default shares it sent on behalf of the honest sender and $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}\}_{j \in T}$. In this way, Sim^r_Φ obtains the purported shares, denoted by $\{\phi'_j : j \in T\}$, that the corrupted servers will send if they were following the protocol. We remark that the corrupted servers may not follow the protocol as the adversary $\mathcal{A}$ can do whatever it wants on the corrupted servers. In case $\mathcal{A}$ decides to behave semi-honestly on the corrupted servers, then $\{\phi'_j : j \in T\}$ would be the expected output.

   Recall that $\overline{o}' = \overline{o}'_1 || \ldots || \overline{o}'_{N'}$. Conditioned on fixing the purported second-round messages from corrupted servers $\{\phi'_j : j \in T\}$, Sim^r_Φ secret shares each of $\overline{o}'_i$ for $i \in [N']$ by calling $\mathsf{Share}_{(4t+3\ell,\ell,m)}$. As a result, Sim^r_Φ obtains $\{\phi_j : j \in [m] \setminus T\}$, which is supposed to be the output of the honest servers. Then, Sim^r_Φ sends $\{\phi'_j : j \in [m] \setminus T\}$ on behalf of the honest servers to $\mathcal{A}$.

5. It outputs whatever $\mathcal{A}$ outputs. The honest sender client does not output.

Proof. By definition of malicious security, we consider the joint output of the sender and the receiver. Since the honest sender does not output, it suffices to show the view of the adversary $\mathcal{A}$ is identically distributed between the real execution and the ideal execution.

Consider a sequence of hybrids starting from the real execution and ending with the ideal execution that corresponds to the simulator Sim^r_Φ:

1. $\underline{\mathsf{Hyb}_0}$: This corresponds to the output of the real execution.
2. $\underline{\mathsf{Hyb}_1}$: In this hybrid, we do the following:
   (a) Based on the shares sent from $\mathcal{A}$ to the honest servers, we extract the receiver's purported input $\overline{x}$ as described in the simulation and sends $\overline{x}$ to the ideal functionality. Note that the honest sender also sends its input $\overline{y}$ to the ideal functionality.
   (b) We obtain the output, which is the evaluations of the polynomials $\overline{p} = (p_1, \ldots, p_N)$, from the ideal functionality. We sample the vector $\overline{o}' = \overline{f}'_1 || \ldots || \overline{f}'_N$ and re-partition it as $\overline{o}' = \overline{o}'_1 || \ldots || \overline{o}'_{N'}$ as described in the simulation.
   (c) Based on the first-round messages $\mathcal{A}$ sent to the honest servers, we use $\mathsf{Extrapolate}$ to compute the purported shares sent to the corrupted servers in T. Using these extrapolated shares and the shares received from the honest sender, we compute the shares that will be sent from the corrupted servers as if they follow the protocol.
   (d) Conditioned on the fixing the above computed shares above, we sample the honest servers' shares of $\overline{o}'_{N'}$ for $i \in [N']$ using $\mathsf{Share}_{(4t+3\ell,\ell,m)}$. Then, we send these shares as the honest server's second-round messages to $\mathcal{A}$.

   The difference from the previous hybrid is that Hyb_0 (the real execution) lets the honest servers compute and send the shares to $\mathcal{A}$, while Hyb_1 simulates the shares and sends them on behalf of the honest servers to $\mathcal{A}$. The admissible adversary following the linear constraints $\mathcal{R}_x$ guarantees that the recovered input $\overline{x}$ from the corrupted receiver is unique. This means that in the real execution Hyb_0, the input $\overline{x}$ inferred from the shares sent to the honest servers is identical to the input $\overline{x}$ extracted by the simulator in Hyb_1. Note that the honest sender has the same input $\overline{y}$ in both hybrids. Therefore, the evaluations of the polynomials $\overline{p} = (p_1, \ldots, p_N)$ inferred from the shares computed by the honest servers in Hyb_0 is identical to the evaluations of the polynomials returned by the ideal functionality in Hyb_1.

   In the real execution Hyb_0, consider the vector $\overline{o} = \overline{p} + \overline{p}^0$, where $\overline{p}^0$ adds a random field element to each monomial of the polynomial p_i under the constraint that the sum of the field elements for p_i is 0. In Hyb_1, we sample random field elements as the value of monomials for each polynomial in $\overline{o}'$ under the constraint that the sum of these field elements equals the evaluation of the polynomial returned by the ideal functionality. We claim that $\overline{o} = \overline{p} + \overline{p}^0$ has the same distribution as $\overline{o}'$:

   - If we compare $\overline{o} = (\overline{p} + \overline{p}^0)$ and $\overline{o}'$ entry by entry, every entry in $\overline{o}$ is a monomial in $\overline{p}$ masked by a random element in $\overline{p}^0$, which has the same distribution as the corresponding random field element in $\overline{o}'$.
   - If we compare $\overline{o}$ and $\overline{o}'$ polynomial by polynomial (corresponding to $p_1, \ldots, p_N$). For every group of entries that form a polynomial $\overline{f}'_i$ in $\overline{o}'$, the sum of these entries is equal to the evaluation of the polynomial in $\overline{p}$ in both hybrids. Thus, they are identically distributed at the polynomial level.

Since the secrets $\overline{o}, \overline{o}'$ to be shared are identically distributed between Hyb_0 and Hyb_1, and the secrets are shared using $\mathsf{Share}_{(4t+3\ell,\ell,m)}$ in both hybrids, the shares sent by the honest servers to $\mathcal{A}$ in Hyb_0 are identically distributed as the shares sent by the simulator on behalf of the honest servers to $\mathcal{A}$ in Hyb_1. In case the adversary $\mathcal{A}$ follows the protocol and controls the corrupted servers to evaluate honestly, the simulator samples the outputting shares conditioned on the corrupted servers following the protocol. Note that if the adversary $\mathcal{A}$ does not let the corrupted servers follow the protocol, then the second-round messages of the corrupted servers do not need to be consistent with the honest servers.

Hence, the view of $\mathcal{A}$ is identically distributed between Hyb_0 and Hyb_1. This implies that the output of Hyb_1 has the same distribution as the output of Hyb_0 (the real execution).

3. $\underline{\mathsf{Hyb}_2}$: In this hybrid, we replace the shares (first-round messages) from the honest sender to the corrupted servers to be shares of default value given by the simulator. Correspondingly, when the simulator simulates the corrupted servers, it also uses shares of default values it sent in the first round, instead of using the real sender's first-round messages as in Hyb_1.

   The view of $\mathcal{A}$ in Hyb_2 has the same distribution as Hyb_1 due to the perfect privacy of the underlying secret sharing scheme.

Note that Hyb_2 corresponds to the ideal execution. Hence, the output of the ideal execution has the same distribution as the output of the real execution. □

The sender is corrupted. The simulator $\underline{\mathsf{Sim}^s_\Phi}$ works as follows:

1. Sim^s_Φ sends (on behalf of the honest receiver) default random values as purported secret shares to the corrupted servers in T.

2. (Extract Input) Sim^s_Φ receives the first-round messages sent to the honest servers from $\mathcal{A}$. Recall that the shares sent by the admissible adversary to the honest servers are consistent with packed Shamir secret sharings. Using the shares sent to the honest servers, Sim^s_Φ recovers the vectors $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^0$ since the number of honest servers is big enough. Since the vectors $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^0$ shared by the admissible adversary satisfy the linear constraints $\mathcal{R}_y$, a unique $\overline{y}$ can be extracted from these vectors. Thus, Sim^s_Φ extracts the corrupted sender's purported input $\overline{y}$ from these vectors. Finally, Sim^s_Φ sends $\overline{y}$ to the ideal functionality.

   Note that for $\overline{p}^0$, since the admissible adversary $\mathcal{A}$ satisfies the linear constraints $\mathcal{R}_y$, it is a valid additive secret sharing of zeros.

3. Sim^s_Φ outputs whatever $\mathcal{A}$ outputs. Note that in the protocol, the sender does not output, but the adversary can output anything it wants. The honest receiver will output based on the evaluation of polynomials obtained from the ideal functionality, which depends on the input $\overline{y}$ sent by Sim^s_Φ.

Proof. To prove security, we need to show the joint output of the sender and the receiver in the real execution is indistinguishable from that in the ideal execution.

For the sender's output, observe that the sender does not receive any message in the protocol. Also, the simulator outputs whatever $\mathcal{A}$ outputs. Therefore, the output of the sender is whatever $\mathcal{A}$ outputs in both the real execution and the ideal simulation.

For the receiver's output, by the t-error correction property of the packed secret sharing scheme, the recovered vectors $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^0$ are unique. Since the admissible adversary guarantees that the recovered vectors satisfy linear constraint $\mathcal{R}_y$, the input $\overline{y}$ extracted by the simulator is unique. Thus the extracted $\overline{y}$ in the ideal execution is the same as the input inferred from the first-round messages sent by $\mathcal{A}$ to the honest servers in the real execution. As a result, the receiver in the ideal world should output $F(\overline{x}, \overline{y})$. In the real execution, the adversary $\mathcal{A}$ controls the corrupted servers and thus can send arbitrary second-round messages from the corrupted servers. However, by the t-error correction property of the secret sharing scheme, regardless of what second-round messages sent by the corrupted servers, the receiver will recover a unique output from the honest servers' second-round messages, which should be consistent with the evaluations using $\overline{y}$ inferred from the first-round messages sent to the honest servers. Therefore, the output of the receiver is identical between the real execution and the ideal execution.

Hence, the joint output of the sender and the receiver is identically distributed between the real execution and the ideal execution. □

4.2 Protocol for Computing Constant-Degree Polynomials

In this section, we describe the conforming protocol $\Phi_{\mathsf{cons}}(\overline{p}, \overline{x}, \overline{y})$ that computes a vector of constant-degree polynomials, which is a natural extension to the protocol Φ_{quad} in the previous section. We use the same notation as in the previous section, except that now $\overline{p} = (p_1, \ldots, p_N)$ is a vector of constant-degree polynomials over $\mathbb{F}$. Let d denote the maximum degree of these polynomials. The extension is rather straightforward: the decomposition now becomes $\overline{p}^x = \overline{p}^x_1 \odot \cdots \odot \overline{p}^x_d$ and $\overline{p}^y = \overline{p}^y_1 \odot \cdots \odot \overline{p}^y_d$ and the protocol is updated correspondingly.

Decomposition. The decomposition rule follows from the previous section. Since the maximum degree is d, we decompose $\overline{p}$ into receiver's vectors $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ and sender's vectors $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d$. For each $i \in [d]$, the decomposed vector $\overline{p}^x_i$ (resp. $\overline{p}^y_i$) contains entries that are either a single variable in $\overline{x}$ (resp. $\overline{y}$) or a field constant. These vectors need to satisfy $\overline{p} = \overline{p}^x \odot \overline{p}^y$, $\overline{p}^x = \overline{p}^x_1 \odot \ldots \odot \overline{p}^x_d$, and $\overline{p}^y = \overline{p}^y_1 \odot \ldots \odot \overline{p}^y_d$. Fix a unique way of decomposition. For example, for a degree $k \leq d$ monomial in $\overline{p}^x$ (resp. $\overline{p}^y$), each variable in the monomial will be assigned to the first k vectors $\overline{p}^x_1, \ldots, \overline{p}^x_k$ lexicographically. Therefore, the decomposition can be expressed as

$$\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d, \overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d \leftarrow \mathsf{Decomp}(\overline{p}).$$

Linear Constraints. Correspondingly, the linear constraints $\mathcal{R}_x$ (resp. $\mathcal{R}_y$) is now over vectors $\overline{p}^x_1, \ldots, \overline{p}^x_d$ (resp. $\overline{p}^0, \overline{p}^y_1, \ldots, \overline{p}^y_d$). Let $\mathcal{R} = (\mathcal{R}_x, \mathcal{R}_y)$ be the set of linear constraints derived from the decomposition.

Admissible Adversary. Let $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ (resp. $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d, \overline{p}^0$) be the vectors to be packed secret shared by the receiver (resp. sender) as a result of the decomposition described above. Recall that m is the number of servers. For each $j \in [m]$, let $\hat{x}_j, \hat{x}_{j,1}, \ldots, \hat{x}_{j,d}$ (resp. $\hat{y}_j, \hat{y}_{j,1}, \ldots, \hat{y}_{j,d}, \hat{0}_j$) be the first-round messages sent by the receiver (resp. sender) to server j, which are the purported shares for $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ (resp. $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d, \overline{p}^0$) respectively.

Definition 4.6 (Admissible Adversary)**.** *An adversary $\mathcal{A}$ corrupting either client and a set of servers T ($|T| \leq t$) is said to be* admissible *w.r.t. linear constraints $\mathcal{R} = (\mathcal{R}_x, \mathcal{R}_y)$ if*

1. *(Consistency) The first-round messages $\hat{x}_j, \hat{x}_{j,1}, \ldots, \hat{x}_{j,d}$ sent by the receiver (resp. $\hat{y}_j$, $\hat{y}_{j,1}$, ..., $\hat{y}_{j,d}$, $\hat{0}_j$ sent by sender) to the honest servers $j \in [m] \setminus T$ are consistent with packed Shamir secret sharings of the vectors $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ (resp. $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d, \overline{p}^0$) respectively.*

2. *(Decomposability Constraints) The first-round messages $\hat{x}_j, \hat{x}_{j,1}, \ldots, \hat{x}_{j,d}$ sent by the receiver (resp. $\hat{y}_j, \hat{y}_{j,1}, \ldots, \hat{y}_{j,d}, \hat{0}_j$ sent by sender) satisfy $\hat{x}_j = \hat{x}_{j,1} \odot \cdots \odot \hat{x}_{j,d}$ (resp. $\hat{y}_j = \hat{y}_{j,1} \odot \cdots \odot \hat{y}_{j,d}$) for all honest servers $j \in [m] \setminus T$.*

3. *(Linear Constraints) Assuming the consistency constraint is satisfied, let $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ (resp. $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d, \overline{p}^0$) be the secret vectors extracted from the first-round messages sent by the receiver (resp. sender) to the honest servers. Then, vectors $\overline{p}^x_1, \ldots, \overline{p}^x_d$ satisfy the linear constraints $\mathcal{R}_x$ (resp. $\overline{p}^y_1, \ldots, \overline{p}^y_d, \overline{p}^0$ satisfy the linear constraints $\mathcal{R}_y$).*

Description of the Protocol. We describe the two-client, m-server, two-round conforming protocol $\Phi_{\mathsf{cons}}(\overline{p}, \overline{x}, \overline{y}) = ((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$ for computing a vector of constant degree polynomials in Fig. 2 below.

Recall that there are m servers, and at most t servers can be corrupted. The final resulting vectors are recovered from a polynomial of degree $2d(t+\ell)$. If we view the sharing as a Reed-Solomon codeword, then the distance of the code is $m - 2d(t+\ell)$. In order to correct t errors, the number of servers m needs to satisfy $t \leq \lfloor (m - 2d(t+\ell) - 1)/2 \rfloor \implies m \geq (2d+2)t + 2d\ell + 1$.

Communication Complexity. Recall that in the protocol, N' is the number of vectors of size ℓ, where $N_{\mathsf{mono}} = N' \cdot \ell$ is the total number of monomials over all polynomials $p_1, \ldots, p_N$ to evaluate. By packed secret sharing, the receiver sends $(d+1)N'$ field elements and the sender sends $(d+2)N'$ field elements to each server. Each server evaluates and then sends N' elements back to the receiver. Therefore, the total number of field elements is $(2d+4)N'm$. Note that $N' = N_{\mathsf{mono}}/\ell$, and thus we have $(2d+4)N'm = (2d+4)mN_{\mathsf{mono}}/\ell$. Each field element takes λ bits. If each polynomial in $p_1, \ldots, p_N$ contains constant number of monomials, then the communication complexity is $O(mN\lambda/\ell)$. Taking $m = O(\ell)$, the communication complexity can be written as $O(N\lambda)$.

Security Proof. We state the security lemma below. The security proof follows identically from the previous section.

Lemma 4.7. *Let ℓ be the packing size of packed secret sharing scheme and let number of servers $m \geq (2d+2)t + 2d\ell + 1$, where t is a small constant fraction of m. The two-client, m-server, two-round conforming protocol Φ_{cons} for computing a vector of constant-degree polynomials is secure against an admissible adversary $\mathcal{A}$ corrupting either client and up to t servers.*

4.3 Protocol for Computing Boolean Circuits

In this section, we describe the conforming protocol $\Phi_{\mathsf{circ}}(C, x, y)$ for computing an arbitrary two-party functionality $f(x, y)$ expressed as a Boolean circuit C. First, we follow the garbling process of the BMR protocol, which reduces the problem of computing circuit C to computing a vector of degree-3 polynomials. Once this reduction is complete, we invoke Φ_{cons} with degree $d = 3$ as a black box to obtain the evaluations of polynomials. The receiver then uses these evaluations to

Protocol Φ_{cons} for Computing Constant-Degree Polynomials:

- **Inputs:** Receiver has input $\overline{x}$ and sender has input $\overline{y}$. The common input $\overline{p} = (p_1, \ldots, p_N)$ is a vector of degree-d polynomials, where d is a constant. Let N_{mono} be the total number of monomials in $p_1, \ldots, p_N$. If N_{mono} is not a multiple of ℓ, we pad $\overline{p}$ with zeros to make N_{mono} divide ℓ.
- **Setup:** Both parties execute $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d, \overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d \leftarrow \mathsf{Decomp}(\overline{p})$.
- **Round 1:**
  The receiver runs $\underline{\mathsf{Share}_1(1^\lambda, \overline{x})}$:
  1. It assigns the corresponding values from $\overline{x}$ for the entries in $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$.
  2. It partitions each of $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ into $N' := N_{\mathsf{mono}}/\ell$ short vectors of size ℓ. Denote these vectors by $\{\overline{v}^x_i : i \in [N']\}$ and $\{\overline{v}^x_{i,1} : i \in [N']\}, \ldots, \{\overline{v}^x_{i,d} : i \in [N']\}$ respectively.
  3. For each $i \in [N']$:
     (a) For each $k \in [d]$, it computes $[\overline{v}^x_{i,k}] = \mathsf{Share}_{(t,\ell,m)}(\overline{v}^x_{i,k})$, where $[\overline{v}^x_{i,k}] := (s_{i,1,k}, \ldots, s_{i,m,k})$.
     (b) It computes $[\overline{v}^x_i] = [\overline{v}^x_{i,1}] \odot \cdots \odot [\overline{v}^x_{i,d}]$, where $[\overline{v}^x_i] := (s_{i,1}, \ldots, s_{i,m})$;
  4. For each $j \in [m]$, let $\hat{x}_j := (s_{1,j}, \ldots, s_{N',j})$, $\hat{x}_{j,1} := (s_{1,j,1}, \ldots, s_{N',j,1})$, $\ldots$, $\hat{x}_{j,d} := (s_{1,j,d}, \ldots, s_{N',j,d})$. It outputs $\{\hat{x}_j, \hat{x}_{j,1}, \ldots, \hat{x}_{j,d}\}_{j \in [m]}$.

  The sender runs $\underline{\mathsf{Share}_2(1^\lambda, \overline{y})}$:
  1. It assigns the corresponding values from $\overline{y}$ for the entries in $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d$.
  2. It samples N vectors $p^0_1, \ldots, p^0_N$ such that for $i \in [N]$, each $p^0_i = (r_{i,1}, \ldots, r_{i,|p_i|})$ contains $|p_i|$ random field elements under the constraint $\sum_{j=1}^{|p_i|} r_{i,j} = 0$. Let $\overline{p}^0 := p^0_1 || \ldots || p^0_N$.
  3. It partitions each of $\overline{p}^0, \overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d$ into $N' := N_{\mathsf{mono}}/\ell$ short vectors similarly as above. Denote these vectors by $\{\overline{v}^0_i : i \in [N']\}$ and $\{\overline{v}^y_i : i \in [N']\}$ and $\{\overline{v}^y_{i,1} : i \in [N']\}, \ldots, \{\overline{v}^y_{i,d} : i \in [N']\}$ respectively.
  4. For each $i \in [N']$, it computes $[\overline{v}^y_{i,1}], \ldots, [\overline{v}^y_{i,d}]$ and $[\overline{v}^y_i]$ similarly as above. Additionally, for each $i \in [N']$, it also computes $[\overline{v}^0_i] = \mathsf{Share}_{((2d+2)t+(2d-1)\ell,\ell,m)}(\overline{v}^0_i)$.
  5. For each $j \in [m]$, define $\hat{0}_j, \hat{y}_j, \hat{y}_{j,1}, \ldots, \hat{y}_{j,d}$ similarly as above. It outputs $\{\hat{y}_j, \hat{y}_{j,1}, \ldots, \hat{y}_{j,d}, \hat{0}_j\}_{j \in [m]}$.
- **Round 2:** Server j runs $\underline{\mathsf{Eval}(1^\lambda, j, (\hat{x}_j, \hat{x}_{j,1}, \ldots, \hat{x}_{j,d}), (\hat{y}_j, \hat{y}_{j,1}, \ldots, \hat{y}_{j,d}, \hat{0}_j))}$ for $j \in [m]$:
  1. **Verify:** It verifies that $\hat{x}_j = \hat{x}_{j,1} \odot \cdots \odot \hat{x}_{j,d}$ and $\hat{y}_j = \hat{y}_{j,1} \odot \cdots \odot \hat{y}_{j,d}$. Note that this will be eventually replaced by the watchlist in the compiler.
  2. **Eval:** It outputs $\phi_j := \hat{x}_j \odot \hat{y}_j + \hat{0}_j$ via N' parallel OLE operations.
- **Output:** The receiver runs $\underline{\mathsf{Dec}(1^\lambda, (\phi_1, \ldots, \phi_m))}$:
  1. It computes $\overline{o} = \mathsf{Rec}_{((2d+2)t+(2d-1)\ell,\ell,m)}(\phi_1, \ldots, \phi_m)$, where $\overline{o} := \overline{o}_1 || \ldots || \overline{o}_{N'}$. For each $i \in [N']$, $\overline{o}_i$ is a vector of ℓ elements.
  2. Note that $\overline{o}$ is a vector of $N_{\mathsf{mono}} = N' \cdot \ell$ elements. It re-partitions monomials in $\overline{o}$ back into N vectors of size $|p_1|, \ldots, |p_N|$ in order. Denote the re-partitions by $\overline{o} = \overline{f}_1 || \ldots || \overline{f}_N$, where $\overline{f}_i = (f_{i,1}, \ldots, f_{i,|p_i|})$, which corresponds to evaluations of monomials in p_i for $i \in [N]$.
  3. For each $i \in [N]$, it computes $z_i = \sum_{j=1}^{|p_i|} f_{i,j}$.
  4. It outputs $\overline{z} := (z_1, \ldots, z_N)$.

Figure 2: Protocol for Computing Constant-Degree Polynomials

carry out the BMR evaluation process and obtain the final output $C(x, y)$. This is the final outer protocol that will be used in our compiler.

Notations. Let C be a Boolean circuit that takes input $x := (x_1, \ldots, x_n) \in \{0,1\}^n$ and input $y := (y_1, \ldots, y_n) \in \{0,1\}^n$, where x_i (resp. y_i) for $i \in [n]$ denote the i-th bit of x (resp. y). The circuit C consists of XOR and AND gates with fan-in 2. Let n_{out} be the number of output wires in C. Note that any two-party function $f(x, y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n_{\mathsf{out}}}$ can be represented by the Boolean circuit C.

Let $|C|$ denote the number of gates in C. For each gate in C, we use g to denote the gate index and use i and j to denote the gate input wires and use k to denote the gate output wire. Suppose the size of the Boolean circuit C is polynomial in n.

Let $\mathcal{F} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a pseudorandom function (PRF). Let Enc be an encryption scheme based on $\mathcal{F}$. Let MAC be a strongly-unforgeable one-time message authentication code scheme. Concretely, for each gate g, the MAC uses a pair of values $(\mathtt{key}_g, \mathtt{b}_g)$ where $\mathtt{key}_g \in \{0,1\}^\lambda \setminus \{0^\lambda\}$ and $\mathtt{b}_g \in \{0,1\}^\lambda$ to authenticate a single bit $z \in \{0,1\}$ by computing $\mathtt{key}_g \cdot z + \mathtt{b}_g$.

4.3.1 Reduction to Computing Degree-3 Polynomials

We first reduce the problem of securely computing a Boolean circuit to securely computing a collection of degree-3 (randomized) polynomials. Familiar readers can recall the *randomizing polynomials* technique introduced by Ishai and Kushilevitz [IK00]. In this paper, we use a variant of the authenticated two-party BMR protocol [BMR90] to concretely instantiate the randomizing polynomials. Note that the BMR protocol garbles all the gates simultaneously, which is suitable for our goal of constructing a two-round protocol.

Inputs. Recall that in BMR garbling, each party's input consists of circuit input bits and λ-bit strings. Since we are reducing to computing polynomials, every input will be converted to field elements of $\mathbb{F}$ before the secret sharing step in the protocol we will describe below. The inputs of BMR protocol consist of the following parts:

1. The receiver's (resp. sender's) masking bit $r_w \in \{0,1\}$ (resp. $s_w \in \{0,1\}$) sampled uniformly at random for each wire w in the circuit. If w is a circuit input wire corresponding to the sender's input, then the receiver's masking bit $r_w = 0$. On the other hand, if w is a circuit input wire corresponding to the receiver's input, then the sender's masking bit $s_w = 0$. For any circuit output wire w, we set both masking bits $r_w = s_w = 0$. In total, there are $O(|C|)$ wires.

2. The receiver's MAC key denoted $\mathtt{key}_g \in \{0,1\}^\lambda \setminus \{0^\lambda\}, \mathtt{b}_g \in \{0,1\}^\lambda$ sampled uniformly at random independently for each gate g in the circuit. The MAC is computed as $\mathsf{MAC}(x) = \mathtt{key}_g \cdot x + \mathtt{b}_g$. There are $|C|$ gates.

3. The sender's two labels $\mathtt{lab}_{w,0}, \mathtt{lab}_{w,1} \in \{0,1\}^\lambda$ sampled uniformly at random for each wire w. If w is the circuit output wire, then we set $\mathtt{lab}_{w,0} = \mathtt{lab}_{w,1} = 0^\lambda$.

4. The receiver's input $x \in \{0,1\}^n$ and the sender's input $y \in \{0,1\}^n$ to the circuit. For each circuit input wire w, let x_w (resp. y_w) denote a single bit of the receiver's (resp. sender's) circuit input. In the BMR protocol, each party sends a masked value (also called a colored

bit) that is computed based on x_w (resp. y_w) and the masking bits. We will describe the computation for masked input values in detail below.

In summary, the receiver's input to the BMR garbled circuit consists of its masking bits $\{r_w : w \text{ is a wire in } C\}$, MAC keys $\{\mathtt{key}_g, \mathtt{b}_g : g \text{ is a gate in } C\}$ and its input string $x \in \{0,1\}^n$; and the sender's input consists of its masking bits $\{s_w : w \text{ is a wire in } C\}$, wire labels $\{(\mathtt{lab}_{w,0}, \mathtt{lab}_{w,1}) : w \text{ is a wire in } C\}$, and its input string $y \in \{0,1\}^n$.

Computing Garbled Gates. We illustrate how to compute a single garbled gate, and all other gates are computed similarly. For every gate g described by the 3-tuple (i, j, k), let r_i, r_j, r_k (resp. s_i, s_j, s_k) be the receiver's (resp. sender's) masking bits for wire i, j, k, respectively, and let (α, β) be the masked input values for g. For each $(\alpha, \beta) \in \{(0,0), (0,1), (1,0), (1,1)\}$, the garbling process works as follows:

Step 1. Compute the gate output bit

$$z_{\alpha,\beta} = g\left((r_i \oplus s_i) \oplus \alpha, (r_j \oplus s_j) \oplus \beta\right) \oplus (r_k \oplus s_k), \tag{1}$$

where we abuse the notation g to also denote the gate type AND/XOR. Our protocol supports free-XOR optimization so we mainly consider AND gates. Denote the arithmetization of the above expression by $\tilde{z}_{\alpha,\beta}$. Observe that $\tilde{z}_{\alpha,\beta}$ is a degree-2 polynomial containing constant number (≤ 11) of monomials.

Step 2. Select the label corresponding to the gate output wire k. This can be expressed as

$$\mathtt{lab}_{k,z_{\alpha,\beta}} = \mathtt{lab}_{k,0} \oplus z_{\alpha,\beta} \cdot (\mathtt{lab}_{k,1} \oplus \mathtt{lab}_{k,0}), \tag{2}$$

which computes a degree-3 polynomial once we expand the computation for $\tilde{z}_{\alpha,\beta}$. Note that the sender can directly share $(\mathtt{lab}_{k,1} \oplus \mathtt{lab}_{k,0}), \mathtt{lab}_{k,0}$ instead of $\mathtt{lab}_{k,1}, \mathtt{lab}_{k,0}$. Thus, it takes at most 12 monomials to compute $\mathtt{lab}_{k,z_{\alpha,\beta}}$ once we expand $\tilde{z}_{\alpha,\beta}$ as an arithmetic expression.

Step 3. Apply the MAC to the gate output bit $z_{\alpha,\beta}$, and denote the result by $\mathtt{mac}_{g,\alpha,\beta}$. The MAC code is computed by

$$\mathtt{mac}_{g,\alpha,\beta} = \mathtt{key}_g \cdot z_{\alpha,\beta} + \mathtt{b}_g.$$

The MAC computation takes at most 12 monomials and it computes a degree-3 polynomial. Note that $z_{\alpha,\beta}$ can be extracted from $\mathtt{mac}_{g,\alpha,\beta}$ given $\mathtt{key}_g, \mathtt{b}_g$ as follows

$$z_{\alpha,\beta} = \begin{cases} 0, & \text{if } \mathtt{mac}_{g,\alpha,\beta} = \mathtt{b}_g, \\ 1, & \text{if } \mathtt{mac}_{g,\alpha,\beta} = \mathtt{key}_g + \mathtt{b}_g. \end{cases}$$

Let $M_{\alpha,\beta}$ be the augmented functionality consisting of the MAC code of the gate output bit $z_{\alpha,\beta}$, and the corresponding label to the gate output bit:

$$M_{\alpha,\beta} = (\mathtt{mac}_{g,\alpha,\beta}, \mathtt{lab}_{k,z_{\alpha,\beta}}). \tag{3}$$

The computation for $M_{\alpha,\beta}$ can be viewed as a degree-3 polynomial to compute $\mathtt{mac}_{g,\alpha,\beta}$, and another degree-3 polynomial to compute $\mathtt{lab}_{k,z_{\alpha,\beta}}$. In total, it takes at most $12 + 12 = 24$ monomials to compute $M_{\alpha,\beta}$.

Step 4. Encrypt $M_{\alpha,\beta}$ with a PRF-based encryption scheme Enc. Note that $M_{\alpha,\beta}$ consists of the λ-bit MAC code of the gate output bit, and the λ-bit gate output label. Recall $\mathcal{F} : \{0,1\}^* \to$

$\{0,1\}^{2\lambda}$ is a PRF. The sender locally evaluates $\mathcal{F}(\mathtt{lab}_{i,\alpha}, g, \beta) \oplus \mathcal{F}(\mathtt{lab}_{j,\beta}, g, \alpha)$ for gate g in C, and thus obtain a 2λ-bit result: the first λ bits are used to one-time pad with the MAC code; the last λ bits are used to one-time pad with the label $\mathtt{lab}_{k,z_{\alpha,\beta}}$. We need the gate index to prevent that the gate that output wire i or j has multiple output wires. Since the PRF is computed locally, we can replace it with a PRG, which can be constructed from a random oracle.

Applying this PRF-based encryption scheme is essentially adding an extra monomial at the end of each polynomial that computes $\mathtt{mac}_{g,\alpha,\beta}$ and $\mathtt{lab}_{k,z_{\alpha,\beta}}$. Thus, the garbled entry $g_{\alpha,\beta} = \mathsf{Enc}(\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}, M_{\alpha,\beta})$ is computed by

$$g_{\alpha,\beta} = M_{\alpha,\beta} \oplus (\mathcal{F}(\mathtt{lab}_{i,\alpha}, g, \beta) \oplus \mathcal{F}(\mathtt{lab}_{j,\beta}, g, \alpha)), \tag{4}$$

which contains two degree-3 polynomials to compute inherited from $M_{\alpha,\beta}$. In total, there are at most $24 + 2 = 26$ monomials to compute $g_{\alpha,\beta}$. In summary, each gate g will have the following garbled table:

$$\begin{aligned}
g_{0,0} &= M_{0,0} \oplus \mathcal{F}(\mathtt{lab}_{i,0}, g, 0) \oplus \mathcal{F}(\mathtt{lab}_{j,0}, g, 0);\\
g_{0,1} &= M_{0,1} \oplus \mathcal{F}(\mathtt{lab}_{i,0}, g, 1) \oplus \mathcal{F}(\mathtt{lab}_{j,1}, g, 0);\\
g_{1,0} &= M_{1,0} \oplus \mathcal{F}(\mathtt{lab}_{i,1}, g, 0) \oplus \mathcal{F}(\mathtt{lab}_{j,0}, g, 1);\\
g_{1,1} &= M_{1,1} \oplus \mathcal{F}(\mathtt{lab}_{i,1}, g, 1) \oplus \mathcal{F}(\mathtt{lab}_{j,1}, g, 1).
\end{aligned}$$

Note that the MAC is to prevent the malicious sender from using a different value for $\mathcal{F}(\mathtt{lab}_{i,\alpha}, g, \beta) \oplus \mathcal{F}(\mathtt{lab}_{j,\beta}, g, \alpha)$. Recall that $M_{\alpha,\beta} = (\mathtt{mac}_{g,\alpha,\beta}, \mathtt{lab}_{k,z_{\alpha,\beta}})$. If the $\mathtt{mac}_{g,\alpha,\beta}$ part is affected by altered PRF output, then the receiver will catch it and abort. If the $\mathtt{lab}_{k,z_{\alpha,\beta}}$ part is affected by the altered PRF output, then the receiver will obtain an altered label. As a result, when decrypting the next layer gate g', the $\mathtt{mac}_{g',\alpha,\beta}$ part will be altered with overwhelming probability and thus get caught.

Circuit Evaluation. This is the same circuit evaluation process as in the BMR protocol. Note that each circuit input wire corresponds to either the receiver's input bit or the sender's input bit. For every input layer gate g described by a 3-tuple (i, j, k), without loss of generality, let α_g associate with wire i (the left input) and let β_g associate with wire j (the right input). We use x_i, y_i (resp. x_j, y_j) to denote the receiver's and the sender's input bit at wire i (resp. j) respectively. If wire i corresponds to the receiver's input bit, then the receiver sets the masked input bit $\alpha_g = x_i \oplus r_i$. Otherwise, the sender sets the masked input bit $\alpha_g = y_i \oplus s_i$. If wire j corresponds to the receiver's input bit, then the receiver sets the masked input $\beta_g = x_j \oplus r_j$. Otherwise, the sender sets the masked input $\beta_g = y_j \oplus s_j$. The invariant we want to maintain throughout the circuit evaluation is

$$z_{\alpha,\beta} = g((r_i \oplus s_i) \oplus \alpha, (r_j \oplus s_j) \oplus \beta) \oplus (r_k \oplus s_k) = g(x_i, y_j) \oplus (r_k \oplus s_k).$$

Note that the invariant holds true for the input layer gates:

$$z_{\alpha,\beta} = g(r_i \oplus \alpha_g, s_j \oplus \beta_g) \oplus (r_k \oplus s_k) = g(x_i, y_j) \oplus (r_k \oplus s_k).$$

The garbling for input layer works as follows. First, the sender and the receiver compute α_g and β_g as a degree-1 polynomial:

$$\begin{aligned}\alpha_g &= \begin{cases} x_i \oplus r_i, & i \text{ is receiver's input wire} \\ y_i \oplus s_i, & i \text{ is sender's input wire} \end{cases}, \\ \beta_g &= \begin{cases} x_j \oplus r_j, & j \text{ is receiver's input wire} \\ y_j \oplus s_j, & j \text{ is sender's input wire} \end{cases}. \end{aligned} \tag{5}$$

Next, we compute the label $\mathtt{lab}_{i,\alpha}$ associated with α_g and $\mathtt{lab}_{j,\beta}$ associated with β_g. The labels of input wires can be computed by degree-2 polynomials as follows:

$$\begin{aligned}\mathtt{lab}_{i,\alpha} &= (1-\alpha_g)\mathtt{lab}_{i,0} + \alpha_g \mathtt{lab}_{i,1} = \mathtt{lab}_{i,0} + \alpha_g(\mathtt{lab}_{i,1} - \mathtt{lab}_{i,0}), \\ \mathtt{lab}_{j,\beta} &= (1-\beta_g)\mathtt{lab}_{j,0} + \beta_g \mathtt{lab}_{j,1} = \mathtt{lab}_{j,0} + \beta_g(\mathtt{lab}_{j,1} - \mathtt{lab}_{j,0}). \end{aligned} \tag{6}$$

For a finite field $\mathbb{F}$ of order 2^λ, we have $\mathtt{lab}_{i,1} - \mathtt{lab}_{i,0} = \mathtt{lab}_{i,1} \oplus \mathtt{lab}_{i,0}$ and $\mathtt{lab}_{j,1} - \mathtt{lab}_{j,0} = \mathtt{lab}_{j,1} \oplus \mathtt{lab}_{j,0}$. Thus, the sender provides $(\mathtt{lab}_{i,0}, (\mathtt{lab}_{i,1} \oplus \mathtt{lab}_{i,0}))$ (resp. $(\mathtt{lab}_{j,0}, (\mathtt{lab}_{j,1} \oplus \mathtt{lab}_{j,0}))$), and if the wire corresponds to the sender's input, it provides degree-1 polynomial $y_i \oplus s_i$ (or $y_j \oplus s_j$). Similarly, if the wire corresponds to the receiver's input, the receiver provides degree-1 polynomial $x_i \oplus r_i$ (or $x_j \oplus r_j$). Then, the sender and the receiver jointly compute the degree-2 polynomial as expressed in equation (6). Then, we compute the MAC for α_g, β_g by degree-2 polynomials:

$$\begin{aligned}\mathtt{mac}_{g,\alpha} &= \mathtt{key}_g \cdot \alpha_g + \mathtt{b}_g, \\ \mathtt{mac}_{g,\beta} &= \mathtt{key}_g \cdot \beta_g + \mathtt{b}_g. \end{aligned} \tag{7}$$

For an input layer gate g, let I_g denote the augmented functionality consisting of $\mathtt{mac}_{g,\alpha}, \mathtt{mac}_{g,\beta}$ and $\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}$:

$$I_g = (\mathtt{mac}_{g,\alpha}, \mathtt{mac}_{g,\beta}, \mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}). \tag{8}$$

In the evaluation phase, once I_g is computed, the receiver obtains $\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}$ and the value of α_g, β_g from $\mathtt{mac}_{g,\alpha}, \mathtt{mac}_{g,\beta}$ since the receiver knows $\mathtt{key}_g$ and $\mathtt{b}_g$. Then, the receiver uses $\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}$ as the PRF key to decrypt the corresponding garbled gate $g_{\alpha,\beta}$ to obtain $M_{\alpha,\beta}$. Next, since the receiver generated the MAC key $\mathtt{key}_g, \mathtt{b}_g$ for all gates, it verifies and obtains the gate output bit $z_{\alpha,\beta}$ from the MAC code. The receiver also decrypts the gate output label $\mathtt{lab}_{k,z_{\alpha,\beta}}$ from $M_{\alpha,\beta}$. This allows the receiver to decrypt the next layer of garbled gates by repeating the same process. The gate evaluation process propagates up layer by layer. In the end, the top output layer gate output $z_{\alpha,\beta}$ is computed. Note that for the circuit output wire, both masking bits are set to 0.

Polynomials to Compute. In summary, we have reduced the problem of securely computing a Boolean circuit C to computing:

1. A collection of degree-3 polynomials over $\mathbb{F}$ that computes the garbled gates

$$\mathcal{G} = \{g_{\alpha,\beta} : (\alpha,\beta) \in \{0,1\}^2 \text{ and } g_{\alpha,\beta} \text{ is a garbled entry for gate } g\}. \tag{9}$$

   Each $g_{\alpha,\beta}$ contains two degree-3 polynomials. Each polynomial contains a constant number of monomials. Each gate has 4 garbled entries. Therefore, there are $O(|C|)$ total monomials to compute in $\mathcal{G}$.

2. A collection of degree-2 polynomials over $\mathbb{F}$ that computes the inputs to the garbled circuit

$$\mathcal{I} = \{I_g : I_g \text{ computes the BMR circuit inputs to an input gate } g\}. \tag{10}$$

Each I_g contains four polynomials. Each polynomial contains a constant number of monomials. Therefore, there are $O(|C'|)$ polynomials to compute, where $|C'| \leq |C|$ is the number of input layer gates.

Hence, securely computing a Boolean circuit C has been reduced to computing all the polynomials in $\mathcal{G} \cup \mathcal{I}$.

Let's summarize the reduction described above. Originally the circuit inputs are x from the receiver and y from the sender. The BMR garbling process makes the sender and the receiver generate extra inputs such as the masking bits, the MAC keys and the wire labels. Therefore, after the reduction, we denote the receiver's input by a vector $\overline{x}$ and the sender's input by a vector $\overline{y}$ over $\mathbb{F}$ where

- $\overline{x}$ consists of
  - circuit input bits $\{x_w : w \text{ is a receiver's circuit input wire}\}$;
  - masking bits $\{r_w : w \text{ is a wire in } C\}$;
  - MAC keys $\{(\mathtt{key}_g, \mathtt{b}_g) : g \text{ is a gate in } C\}$.
- $\overline{y}$ consists of
  - circuit input bits $\{y_w : w \text{ is a sender's circuit input wire}\}$;
  - masking bits $\{s_w : w \text{ is a wire in } C\}$;
  - wire labels $\{(\mathtt{lab}_{w,0}, (\mathtt{lab}_{w,1} \oplus \mathtt{lab}_{w,0})) : w \text{ is a wire in } C\}$;
  - precomputed PRF evaluations $(\mathcal{F}(\mathtt{lab}_{i,\alpha}, g, \beta) \oplus \mathcal{F}(\mathtt{lab}_{j,\beta}, g, \alpha))$ for each $(\alpha, \beta) \in \{0,1\}^2$ for each gate g (then chopped into elements of size λ).

Note that $\overline{x}, \overline{y}$ contain $O(|C|)$ field elements of $\mathbb{F}$. Let $\overline{p} := \mathcal{G}||\mathcal{I}$ be a vector of polynomials of degree at most 3. We denote the reduction from circuit C with input x, y to degree-3 polynomials as

$$\overline{p}, \overline{x}, \overline{y} \leftarrow \mathsf{Reduc}(C, x, y).$$

4.3.2 Description of the Protocol

We start with the definition of the admissible adversary. The admissible adversary $\mathcal{A}$ builds on top of the admissible adversary in the protocol Φ_{cons} for computing constant-degree polynomials with maximum degree $d = 3$. Additionally, the reduction process above converts Boolean values such as the circuit input bits, masking bits into field elements of $\mathbb{F}$. Therefore, we introduce Boolean constraints, denoted by $\mathcal{B} = (\mathcal{B}_x, \mathcal{B}_y)$ that ensure the corresponding positions in the vectors $\overline{p}^x, \overline{p}^x_1, \ldots, \overline{p}^x_d$ (resp. $\overline{p}^y, \overline{p}^y_1, \ldots, \overline{p}^y_d$) that correspond to input bits or masking bits to have values 0 or 1 in $\mathbb{F}$.

Definition 4.8 (Admissible Adversary)**.** *An adversary $\mathcal{A}$ corrupting either client and a set of servers T ($|T| \leq t$) is said to be* admissible *w.r.t. linear constraints $\mathcal{R} = (\mathcal{R}_x, \mathcal{R}_y)$ and Boolean constraints $\mathcal{B} = (\mathcal{B}_x, \mathcal{B}_y)$ if*

1. *(Consistency Constraints) The first-round messages $\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}$ sent by the receiver (resp. $\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j$ sent by sender) to the honest servers $j \in [m] \setminus T$ are consistent with packed Shamir secret sharings of the vectors $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ (resp. $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3, \overline{p}^0$) respectively.*

2. *(Decomposability Constraints) The first-round messages $\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}$ sent by the receiver (resp. $\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j$ sent by sender) satisfy $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2} \odot \hat{x}_{j,3}$ (resp. $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2} \odot \hat{y}_{j,3}$) for all honest servers $j \in [m] \setminus T$.*

3. *(Linear Constraints) Assuming the consistency constraint is satisfied, let $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ (resp. $\overline{p}^y$, $\overline{p}^y_1$, $\overline{p}^y_2, \overline{p}^y_3$, $\overline{p}^0$) be the secret vectors extracted from the first-round messages sent by the receiver (resp. sender) to the honest servers. Then, $\overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ satisfy the linear constraints $\mathcal{R}_x$ (resp. $\overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3, \overline{p}^0$ satisfy $\mathcal{R}_y$).*

4. *(Boolean Constraints) Assuming the consistency constraint is satisfied, let $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ (resp. $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3, \overline{p}^0$) be the secret vectors extracted from the first-round messages sent by the receiver (resp. sender) to the honest servers. Then, $\overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ satisfy the Boolean constraints $\mathcal{B}_x$ (resp. $\overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$ satisfy $\mathcal{B}_y$).*

The Protocol. In Figure 3 below, we describe the two-round, two-client, m-server conforming protocol $\Phi_{\mathsf{circ}}(C, x, y) := ((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$ for computing a Boolean circuit C. Note that each party does not need to know the other party's actual input values to perform the reduction $\overline{p}, \overline{x}, \overline{y} \leftarrow \mathsf{Reduc}(C, x, y)$, as it treats the other party's input as variables (symbols).

Communication Complexity. There are $|C|$ gates in the circuit. For each garbled gate in $\mathcal{G}$, we need to compute a constant number of polynomials, and each polynomial contains constant number of monomials. For each input layer gate in $\mathcal{I}$, it computes a constant number of polynomials for the masked input values and their labels, and each polynomial contain constant number of monomials. Therefore, the total number of monomials $N_{\mathsf{mono}} = O(|C|)$. Since the packing size for the packed Shamir secret sharing used in the protocol is ℓ, the number of shares is $N' := N_{\mathsf{mono}}/\ell$ per server. Each field element takes λ bits to represent. There are m servers. Hence, the communication complexity is $O(m|C|\lambda/\ell)$. If we take $m = O(\ell)$, then the communication complexity would be $O(|C|\lambda)$.

4.3.3 Security Proof

We state the security theorem below.

Theorem 4.9. *Let ℓ be the packing size of packed secret sharing scheme, and let number of servers $m \geq 8t + 6\ell + 1$, where t is a small constant fraction of m. The two-client, m-server, two-round conforming protocol Φ_{circ} for computing a Boolean circuit C is secure against an admissible adversary $\mathcal{A}$ corrupting either client and up to t servers.*

Let $\mathcal{A}$ be an admissible adversary that corrupts either client and up to t servers. Let $\mathsf{Sim}_\Phi = (\mathsf{Sim}^r_\Phi, \mathsf{Sim}^s_\Phi)$ be the simulator in the ideal world, where Sim^r_Φ (resp. Sim^s_Φ) is the simulator when the adversary $\mathcal{A}$ corrupts the receiver (resp. sender) and a subset of $T \subseteq [m]$ ($|T| \leq t$) servers. Let $C(x, y)$ be the ideal functionality that outputs Boolean circuit evaluation on input x, y.

The receiver is corrupted. The simulator $\underline{\mathsf{Sim}^r_\Phi}$ works as follows:

1. It samples the purported input $\overline{y}'$ on behalf of the sender, which consists of sender's masking bits $\{s_w : w \text{ is a wire in } C\}$, wire labels $\{(\mathtt{lab}_{w,0}, (\mathtt{lab}_{w,0} \oplus \mathtt{lab}_{w,1})) : w \text{ is a wire in } C\}$, the

Protocol $\Phi_{\mathsf{circ}}(C, x, y)$ for Computing Boolean Circuits:

- **Inputs:** Receiver has input x and sender has input y. Both parties have the common input Boolean circuit C.
- **Setup:** Both parties first execute the reduction $\overline{p}, \overline{x}, \overline{y} \leftarrow \mathsf{Reduc}(C, x, y)$, where $\overline{p} = (p_1, \ldots, p_N)$ is a vector of degree-3 polynomials. Note that $\overline{p} = \mathcal{G}||\mathcal{I}$ as described in the reduction above, where $\mathcal{G}$ computes all the garbled gates, and $\mathcal{I}$ computes the masked circuit inputs and their labels. Let N_{mono} be the total number of monomials in $\overline{p}$. If N_{mono} is not a multiple of ℓ, we pad $\overline{p}$ with zeros to make N_{mono} divide ℓ. Then, both parties execute the decomposition procedure $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3, \overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3 \leftarrow \mathsf{Decomp}(\overline{p})$.
- **Round 1:**
  The receiver runs $\underline{\mathsf{Share}_1(1^\lambda, \overline{x})}$:
  1. It invokes $\mathsf{Share}_1(1^\lambda, \overline{x})$ in Φ_{cons} with degree $d = 3$. Denote the output by $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}\}_{j\in[m]}$, where $\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3} \in \mathbb{F}^{N'}$ and $N' = N_{\mathsf{mono}}/\ell$. The verification key $vk = \overline{x}$.

  The sender runs $\underline{\mathsf{Share}_2(1^\lambda, \overline{y})}$:
  1. It invokes $\mathsf{Share}_2(1^\lambda, \overline{y})$ in Φ_{cons} with degree $d = 3$. Denote the output by $\{\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j\}_{j\in[m]}$, where $\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j \in \mathbb{F}^{N'}$.
- **Round 2:** Server j runs $\underline{\mathsf{Eval}(1^\lambda, j, (\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}), (\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j))}$ for each $j \in [m]$:
  1. **Verify:** It verifies that $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2} \odot \hat{x}_{j,3}$ and $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2} \odot \hat{y}_{j,3}$. Note that this will be eventually replaced by the watchlist in the compiler.
  2. **Eval:** It outputs $\phi_j := \hat{x}_j \odot \hat{y}_j + \hat{0}_j$ via N' parallel OLE operations. This will be instantiated by the inner protocol.
- **Output:** The receiver runs $\underline{\mathsf{Dec}(1^\lambda, vk, (\phi_1, \ldots, \phi_m))}$:
  1. It invokes $\mathsf{Dec}(1^\lambda, (\phi_1, \ldots, \phi_m))$ in Φ_{cons} with degree $d = 3$. As a result it obtains vector $\overline{z} := (z_1, \ldots, z_N)$ as the evaluations of polynomials $\overline{p} = (p_1, \ldots, p_N)$, which are polynomials in $\mathcal{G} \cup \mathcal{I}$.
  2. (BMR Evaluation) It evaluates the circuit C as follows:
     (a) For each input layer gate g with input wires i, j and output wire k,
        i. It reads off MACs $\mathtt{mac}_{g,\alpha}, \mathtt{mac}_{g,\beta}$ and wire labels $\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}$ from evaluations of polynomials in $\mathcal{I}$.
        ii. It verifies the MAC and obtains the masked circuit inputs α_g, β_g from $\mathtt{mac}_{g,\alpha}, \mathtt{mac}_{g,\beta}$. Abort and output $\perp$ if the MAC check fails.
        iii. Based on α_g, β_g, it reads the garbled entry $g_{\alpha,\beta}$ from $\mathcal{G}$. Then, it uses the labels $\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}$ to decrypt $g_{\alpha,\beta}$ by computing $g_{\alpha,\beta} \oplus \mathcal{F}(\mathtt{lab}_{i,\alpha}, g, \beta_g) \oplus \mathcal{F}(\mathtt{lab}_{j,\beta}, g, \alpha_g)$. As a result, it obtains $M_{\alpha,\beta} = (\mathtt{mac}_{g,\alpha,\beta}, \mathtt{lab}_{k,z_{\alpha,\beta}})$.
        iv. It checks if $\mathtt{mac}_{g,\alpha,\beta}$ is consistent with the MAC key $(\mathtt{key}_g, \mathtt{b}_g)$. If not, abort and output $\perp$. Otherwise, it obtains $z_{\alpha,\beta}$ for gate output wire k and its corresponding label $\mathtt{lab}_{k,z_{\alpha,\beta}}$.
     (b) Now it obtains the gate output bit $z_{\alpha,\beta}$ and its corresponding label $\mathtt{lab}_{k,z_{\alpha,\beta}}$ for all input layer gates. It repeats the same process as described above to evaluate all the non-input-layer gates layer by layer. In the end, it obtains the circuit output and outputs.

Figure 3: Protocol for Computing Boolean Circuits

sender's input string $\{y_w : w \text{ is sender's circuit input wire}\}$, and the precomputed PRF evaluations. It runs $\mathsf{Share}_2(1^\lambda, \overline{y}')$ and sends the first-round messages (on behalf of the honest sender) to the corrupted servers in T.

2. (Extract Input) From the first-round messages sent by $\mathcal{A}$ to the honest servers, it recovers the vectors $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x, \overline{p}_3^x$ shared by $\mathcal{A}$. Note that we are able to extract the secret vectors since the number of corrupted servers t is within the error-correction distance. From $\overline{p}^x, \overline{p}_1^x, \overline{p}_2^x, \overline{p}_3^x$, it extracts a unique input vector $\overline{x}$ that the adversary $\mathcal{A}$ uses. Note that $\overline{x}$ is unique due to the decomposability and linear constraint property of the admissible adversary.

3. (Simulating the Gates) From $\overline{x}$, it extracts the circuit input x used by the adversary $\mathcal{A}$. Then, it sends x to the ideal functionality $C(x, y)$ to obtain the circuit output $z \in \{0,1\}^{n_{\mathsf{out}}}$. Using the simulator for the BMR garbling process, it assigns the gate output bit o_g for every gate g's output wire in the circuit. Next, for each gate g represented by the 3-tuple (i, j, k):

   (a) It computes the masked gate output bit $z_{\alpha,\beta}$ using simulated gate output bit o_g, its own generated masking bit s_k, and the extracted masking bit r_k from $\overline{x}$; it also computes masked circuit input values α_g, β_g.

   (b) It computes the MAC code $\mathtt{mac}_{g,\alpha,\beta}$ of $z_{\alpha,\beta}$ using the MAC keys $\mathtt{key}_g, \mathtt{b}_g$ extracted from $\overline{x}$; it also computes the MAC code $\mathtt{mac}_{g,\alpha}, \mathtt{mac}_{g,\beta}$ for input gates.

   (c) It computes the corresponding gate output label $\mathtt{lab}_{k,z_{\alpha,\beta}}$ using its own generated $(\mathtt{lab}_{k,0}, (\mathtt{lab}_{k,0} \oplus \mathtt{lab}_{k,1}))$; it also computes input wire labels $\mathtt{lab}_{i,\alpha}, \mathtt{lab}_{j,\beta}$ for input-layer gates.

   (d) It computes the local PRF encryption using the gate output label and gate index.

   Then it follows the reduction process to generate the output of the polynomials $\overline{p} = (p_1, \ldots, p_N)$ in $\mathcal{G}||\mathcal{I}$. Using the simulator for Φ_{cons} with degree $d = 3$, it obtains the second-round messages that the servers will send to $\mathcal{A}$. These are packed secret shares of the monomials of $\overline{p}$. Then, it sends the shares on behalf of the honest servers to the adversary $\mathcal{A}$.

4. The simulator outputs whatever $\mathcal{A}$ outputs. The honest sender client does not output.

Proof. Consider the following sequence of hybrids.

1. $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution.

2. $\underline{\mathsf{Hyb}_1}$: In this hybrid, we replace the sender's wire labels and masking bits with the simulated wire labels and masking bits, and update the corresponding polynomials in $\mathcal{G}||\mathcal{I}$ to use the simulated labels and masking bits. Since the labels are sampled uniformly at random in both the real execution and the simulator, Hyb_0 and Hyb_1 are identically distributed.

3. $\underline{\mathsf{Hyb}_2}$: In this hybrid, we replace the real gate output wire values with simulated gate output wire values given by the BMR simulator, and update the polynomials in $\mathcal{G}||\mathcal{I}$ to use the simulated gate output wire values. By the security of the BMR protocol, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_1$.

4. $\underline{\mathsf{Hyb}_3}$: In this hybrid, we replace the shares (first-round messages) from the honest sender client to the corrupted servers to be output of $\mathsf{Share}_2(1^\lambda, \overline{y}')$ generated by the simulator. Correspondingly, when the simulator simulates the corrupted servers, it also uses the simulated

shares it sent in the first round, instead of using the real sender client's first-round messages. The view of $\mathcal{A}$ in Hyb_3 has the same distribution as Hyb_2 due to the perfect privacy of the underlying secret sharing scheme.

Note that Hyb_3 corresponds to the ideal execution. Hence, the output of the ideal execution is computationally indistinguishable from the output of the real execution. □

The sender is corrupted. The simulator $\underline{\mathsf{Sim}^s_\Phi}$ works as follows:

1. It samples the purported input $\overline{x}'$ on behalf of the receiver, which consists of input masking bits $\{r_w : w \text{ is a wire in } C\}$, the MAC keys $\{(\mathtt{key}_g, \mathtt{b}_g) : g \text{ is a gate in } C\}$, and input string $\{x_w : w \text{ is receiver's circuit input wire}\}$. It runs $\mathsf{Share}_1(1^\lambda, \overline{x}')$ and sends the first-round messages on (behalf of the honest receiver) to the corrupted servers in T.

2. (Extract Input) From the first-round messages sent by $\mathcal{A}$ to the honest servers, it extracts the vectors $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3, \overline{p}^0$ shared by $\mathcal{A}$. Note that we are able to recover the secret vectors since the number of corrupted servers t is within the error-correction distance for large enough m.

   From $\overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$, it extracts a unique input vector $\overline{y}$ that the adversary $\mathcal{A}$ uses. Note that $\overline{y}$ is unique due to the decomposability and linear constraint property of the admissible adversary.

3. It emulates the server function using its own generated $\overline{x}'$ and extracted $\overline{y}$ to obtain the evaluation of polynomials in $\mathcal{G}||\mathcal{I}$. Then it checks the MAC code, if the MAC check fails, then it instructs the ideal functionality to output $\perp$. Else, it extracts the input y from $\overline{y}$ and sends y to the ideal functionality $C(x, y)$. The simulator outputs whatever $\mathcal{A}$ outputs. The honest receiver client outputs based on whatever the ideal functionality sends to it.

Proof. Consider the following sequence of hybrids.

1. $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution.

2. $\underline{\mathsf{Hyb}_1}$: In this hybrid, we replace the shares sent by the real receiver client to the corrupted servers with the output of $\mathsf{Share}_1(1^\lambda, \overline{x}')$ generated by the simulator. The view of $\mathcal{A}$ in Hyb_1 has the same distribution as Hyb_0 due to the perfect privacy of the underlying secret sharing scheme.

3. $\underline{\mathsf{Hyb}_2}$: In this hybrid, we extract $\overline{y}$ from the adversary $\mathcal{A}$, and emulates the server function using $\overline{x}'$ and $\overline{y}$ to obtain the polynomial evaluations of $\mathcal{G}||\mathcal{I}$. Then we check the MAC code, if the MAC check fails, we instruct the ideal functionality to output $\perp$. Else, it extracts the input y from $\overline{y}$ and sends y to the ideal functionality $C(x, y)$. It follows from the strong unforgeability of the one-time MAC scheme that $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_1$.

Note that Hyb_2 corresponds to the ideal execution. Following the same analysis as Φ_{cons} (or Φ_{quad}), the output of the ideal execution is indistinguishable from the output of the real execution. □

Folklore protocol $\Pi_{\mathsf{singleOLE}}$:

- **Inputs:** Receiver has input $x \in \mathbb{F}$, and sender has input $a, b \in \mathbb{F}$.
- **Setup:** Assume a preprocessed OLE correlation: $x_r, a_r, b_r \in \mathbb{F}$ are sampled uniformly at random and $y_r = a_r x_r + b_r$. The receiver obtains (x_r, y_r) and the sender obtains (a_r, b_r).
- **Round 1:** $\Pi_{\mathsf{singleOLE},1}(x, x_r)$: The receiver computes $x' = x - x_r$, and sends $\pi_1 = x'$ to the sender.
- **Round 2:** $\Pi_{\mathsf{singleOLE},2}(\pi_1, a, b, a_r, b_r)$: The sender computes $c = ax' + b - b_r$ and $a' = a - a_r$. Then, it sends $\pi_2 = (c, a')$ to the receiver.
- **Output:** $\mathsf{out}_{\mathsf{singleOLE}}(\pi_2, x_r, y_r)$: The receiver computes $y = y_r + c + a' x_r$ and outputs y.

Figure 4: Performing a single OLE using an OLE correlation

5 Inner Protocol from OT Correlations

In the outer protocol Φ_{circ}, the server's evaluation function Eval needs to perform N' parallel Oblivious Linear Evaluation (OLE) operations. Our two-round, semi-honest inner protocol instantiates these OLE computations. Let $\mathbb{F}$ denote the finite field used in both the outer and inner protocol, with field order $|\mathbb{F}| = 2^\lambda$. We first describe a trivial inner protocol, Π_{OLE}, which is based on preprocessed random OLE correlations over $\mathbb{F}$. Then, with the help of *multiplication-friendly embeddings* (MFEs) [CCXY18], we construct a second inner protocol, Π^*_{OLE}, which is based on preprocessed random bit-OT correlations. Both Π_{OLE} and Π^*_{OLE} can be composed with the outer protocol in our compiler.

5.1 Inner Protocol from OLE correlations

Assuming random OLE correlations $((x_r, y_r), (a_r, b_r))$ where $y_r = a_r x_r + b_r$, any arbitrary OLE task (x, a, b) can be computed using a two-round folklore protocol, which we denote by $\Pi_{\mathsf{singleOLE}} := (\Pi_{\mathsf{singleOLE},1}, \Pi_{\mathsf{singleOLE},2}, \mathsf{out}_{\mathsf{singleOLE}})$. More specifically, $\Pi_{\mathsf{singleOLE},1}$ takes receiver's input $x \in \mathbb{F}$ and the correlated randomness $x_r, y_r \in \mathbb{F}$ as input, and outputs the first-round message π_1. $\Pi_{\mathsf{singleOLE},2}$ takes the first-round message π_1, the sender's input $a, b \in \mathbb{F}$, and the correlated randomness $a_r, b_r \in \mathbb{F}$ as input, and outputs the second-round message π_2. Finally, $\mathsf{out}_{\mathsf{singleOLE}}$ takes the second-round message π_2 and receiver's correlated randomness $x_r, y_r \in \mathbb{F}$ as input, and outputs $y = ax + b$. We describe the protocol $\Pi_{\mathsf{singleOLE}}$ in Fig. 4.

The total number of field elements exchanged in $\Pi_{\mathsf{singleOLE}}$ is 3. The correctness of the protocol follows from expression:

$$\begin{aligned} y &= y_r + c + a' x_r \\ &= y_r + (ax' + b - b_r) + (a - a_r)x_r \\ &= y_r + (ax - ax_r + b - b_r) + ax_r - a_r x_r \\ &= y_r - a_r x_r - b_r + ax + b \\ &= ax + b. \end{aligned}$$

The security is rather straightforward: the sender gains no information about receiver's input x because x is masked by a randomly sampled x_r. On the other hand, the receiver gains no information about (a, b) beyond the fact $y = ax + b$ from second-round message (c, a') since $ax' + b$ (which contains information about a and b) is masked by a uniform random b_r, and a is masked

The Inner Protocol $\underline{\Pi_{\mathsf{OLE}} = (\Pi_{\mathsf{OLE},1}, \Pi_{\mathsf{OLE},2}, \mathsf{out}_{\mathsf{OLE}})}$:

- **Inputs:** Receiver has input $\mathbf{x} \in \mathbb{F}^k$, and sender has input $\mathbf{a}, \mathbf{b} \in \mathbb{F}^k$.
- **Setup:** Assume k preprocessed OLE correlations: vectors $\mathbf{x}_r, \mathbf{a}_r, \mathbf{b}_r \in \mathbb{F}^k$ are sampled uniformly at random, and $\mathbf{y}_r = \mathbf{a}_r \odot \mathbf{x}_r + \mathbf{b}_r$. Let $\mathbf{r}_{\mathsf{rec}} = (\mathbf{x}_r, \mathbf{y}_r) \in \mathbb{F}^{2k}$ be the correlated randomness held by the receiver, and let $\mathbf{r}_{\mathsf{sen}} = (\mathbf{a}_r, \mathbf{b}_r) \in \mathbb{F}^{2k}$ be the correlated randomness held by the sender.
- **Round 1:** $\underline{\Pi_{\mathsf{OLE},1}(\mathbf{x}, \mathbf{r}_{\mathsf{rec}})}$: The receiver computes $\mathbf{x}' = \mathbf{x} - \mathbf{x}_r$, and sends $\pi_1 = \mathbf{x}'$ to the sender.
- **Round 2:** $\underline{\Pi_{\mathsf{OLE},2}(\pi_1, \mathbf{a}, \mathbf{b}, \mathbf{r}_{\mathsf{sen}})}$: The sender computes $\mathbf{c} = \mathbf{a} \odot \mathbf{x}' + \mathbf{b} - \mathbf{b}_r$ and $\mathbf{a}' = \mathbf{a} - \mathbf{a}_r$. Then, it sends $\pi_2 = (\mathbf{c}, \mathbf{a}')$ to the receiver.
- **Output:** $\underline{\mathsf{out}_{\mathsf{OLE}}(\pi_2, \mathbf{r}_{\mathsf{rec}})}$: The receiver computes $\mathbf{y} = \mathbf{y}_r + \mathbf{c} + \mathbf{a}' \odot \mathbf{x}_r$. Then, it outputs $\mathbf{y}$.

Figure 5: Description of the Inner Protocol

by uniform random a_r.

Let $k \in \mathbb{N}$. Assuming random OLE correlations, we describe a two-round, semi-honest protocol $\Pi_{\mathsf{OLE}}(\mathbf{x}, \mathbf{a}, \mathbf{b}) := (\Pi_{\mathsf{OLE},1}, \Pi_{\mathsf{OLE},2}, \mathsf{out}_{\mathsf{OLE}})$ that computes k instances of OLE in parallel. Note that Π_{OLE} acts as the inner protocol to perform the Eval function in Φ_{circ}. The protocol Π_{OLE} is described in Fig. 5 below.

The communication complexity of Π_{OLE} is $3k \log |\mathbb{F}|$. Plugging Π_{OLE} in the context of the outer protocol Φ_{circ}, taking $k = N' = N_{\mathsf{mono}}/\ell = O(|C|/\ell)$ and $\log |\mathbb{F}| = \lambda$. Note that each server $j \in [m]$ in the outer protocol runs a copy of Π_{OLE} independently. Therefore, the total communication complexity is $O(m|C|\lambda/\ell)$, which is $O(|C|\lambda)$ when $m = \Theta(\ell)$. The semi-honest security of $\Pi_{\mathsf{OLE}}(\mathbf{x}, \mathbf{a}, \mathbf{b})$ follows directly from the security of the folklore protocol that computes a single OLE from an OLE correlation.

5.2 Reducing OLE correlations to OT correlations

Our inner protocol makes use of preprocessed OLE correlations over $\mathbb{F}$ where $|\mathbb{F}| = 2^\lambda$. With the help of multiplication-friendly embeddings (MFE) [CCXY18], we can reduce a random OLE correlation over $\mathbb{F}_{2^\lambda}$ to $O(\lambda)$ random OT correlations over $\mathbb{F}_2$.

Let $\mathsf{GR}(p^k, s)$ denote the Galois ring over $\mathbb{Z}_{p^k}$ of degree s, which is a ring extension $\mathbb{Z}_{p^k}/(f(X))$ of $\mathbb{Z}_{p^k}$, where $f(X) \in \mathbb{Z}_{p^k}[X]$ is monic polynomial of degree s over $\mathbb{Z}_{p^k}$ whose reduction modulo p is irreducible over $\mathbb{Z}_p$. The following definition is taken from [EXY22].

Definition 5.1 (Multiplication-Friendly Embedding)**.** *Let $r, s \in \mathbb{N}$. A pair of $\mathbb{Z}_{p^k}$-module homomorphisms $\rho : \mathbb{Z}_{p^k}^r \to \mathsf{GR}(p^k, s)$ and $\mu : \mathsf{GR}(p^k, s) \to \mathbb{Z}_{p^k}^r$ is a multiplication-friendly embedding (MFE), if for all $x, y \in \mathsf{GR}(p^k, s)$, it holds that $x \cdot y = \rho(\mu(x) \odot \mu(y))$.*

Note that ρ and μ are both $\mathbb{F}_p$-linear maps. We state the following theorem from [CRX21], which is explicitly stated in [EXY22].

Theorem 5.2. *There exists an explicit MFE family*

$$\left(\rho_s : \mathbb{Z}_{p^k}^{r(s)} \to \mathsf{GR}(p^k, s),\ \mu_s : \mathsf{GR}(p^k, s) \to \mathbb{Z}_{p^k}^{r(s)}\right)_{s \in \mathbb{N}}$$

with $r(s)/s \to 5.12$ as $s \to \infty$.

OLE correlation from OT correlations Π_{fromOT}:

- **Setup:**
  - Fix a MFE $(\rho : \mathbb{F}_2^r \to \mathbb{F}_{2^\lambda},\ \mu : \mathbb{F}_{2^\lambda} \to \mathbb{F}_2^r)$ such that for any $a, x \in \mathbb{F}_{2^\lambda}$, we have $a \cdot x = \rho\big(u_1 v_1, \ldots, u_r v_r\big)$ where $u_i = \mu_i(a)$ (resp. $v_i = \mu_i(x)$) is the output of $\mu(a)$ (resp. $\mu(x)$) projected to the i-th bit.
  - Assume $r = O(\lambda)$ independent random bit-OT correlations are preprocessed: for each $i \in [r]$ the sender holds two bits $(S_i^0, S_i^1) \in \{0,1\}^2$ and the receiver holds $\big(C_i,\ R_i := S_i^{C_i}\big)$ with $C_i \leftarrow \{0,1\}$. Let $r_{\mathsf{rec}} := ((C_1, R_1), \ldots, (C_r, R_r))$ be the receiver's correlated randomness, and let $r_{\mathsf{sen}} := ((S_1^0, S_1^1), \ldots, (S_r^0, S_r^1))$ be the sender's correlated randomness.
- **Local Sampling:** The receiver samples $x_r \leftarrow \mathbb{F}_{2^\lambda}$ and the sender samples $a_r, b_r \leftarrow \mathbb{F}_{2^\lambda}$.
- **Round 1:** The receiver runs $\underline{\Pi_{\mathsf{fromOT},1}(x_r, r_{\mathsf{rec}})}$ defined as follows:
  1. It computes $(v_1, \ldots, v_r) = \mu(x_r)$.
  2. It computes $d_i = v_i \oplus C_i$ for each $i \in [r]$.
  3. It sends $\pi_1 = (d_1, \ldots, d_r)$ to the sender.
- **Round 2:** The sender runs $\underline{\Pi_{\mathsf{fromOT},2}(\pi_1, a_r, b_r, r_{\mathsf{sen}})}$ defined as follows:
  1. It computes $(u_1, \ldots, u_r) = \mu(a_r)$.
  2. For each $i \in [r]$, it samples $t_i \leftarrow \{0,1\}$ uniformly at random, and then computes
  $$e_i^0 = t_i \oplus (u_i \cdot d_i) \oplus S_i^0, \quad e_i^1 = t_i \oplus (u_i \cdot (1 \oplus d_i)) \oplus S_i^1.$$
  3. It computes $T = \rho(t_1, \ldots, t_r) \in \mathbb{F}_{2^\lambda}$. Then it computes the masked field element $Y = T + b_r \in \mathbb{F}_{2^\lambda}$.
  4. It sends $\pi_2 = \big((e_1^0, e_1^1), \ldots, (e_r^0, e_r^1), Y\big)$ to the receiver.
- **Output:** The receiver runs $\underline{\mathsf{out}_{\mathsf{fromOT}}(\pi_2, r_{\mathsf{rec}})}$ defined as follows:
  1. For each $i \in [r]$, using its stored (C_i, R_i), it computes
  $$o_i = e_i^{C_i} \oplus R_i = \begin{cases} t_i \oplus (u_i \cdot d_i) \oplus S_i^0 \oplus S_i^0, & C_i = 0, \\ t_i \oplus (u_i \cdot (1 \oplus d_i)) \oplus S_i^1 \oplus S_i^1, & C_i = 1, \end{cases} = t_i \oplus (u_i v_i).$$
  2. It computes $R = \rho(o_1, \ldots, o_r) \in \mathbb{F}_{2^\lambda}$ and $y_r = R + Y = \rho(u_1 v_1, \ldots, u_r v_r) + b_r = a_r x_r + b_r$. It outputs y_r.

In the end, the receiver obtains (x_r, y_r), and the sender already holds (a_r, b_r), and $y_r = a_r x_r + b_r$.

Figure 6: Obtaining an OLE correlation from random OT correlations and MFE

Taking $p = 2, k = 1, s = \lambda$, then Galois ring $\mathsf{GR}(p^k, s)$ is equivalent to finite field $\mathbb{F}_{2^\lambda}$. Note that ring $\mathbb{Z}_2$ is equivalent to finite field $\mathbb{F}_2$. Thus, we have that there exists an explicit MFE family

$$\Big(\rho_\lambda : \mathbb{F}_2^{r(\lambda)} \to \mathbb{F}_{2^\lambda},\ \mu_\lambda : \mathbb{F}_{2^\lambda} \to \mathbb{F}_2^{r(\lambda)}\Big)_{\lambda \in \mathbb{N}}$$

with $r(\lambda)/\lambda \to 5.12$ as $\lambda \to \infty$.

Using Theorem 5.2 with above parameters, we construct a two-round, semi-honest protocol $\Pi_{\mathsf{fromOT}} := (\Pi_{\mathsf{fromOT},1}, \Pi_{\mathsf{fromOT},2}, \mathsf{out}_{\mathsf{fromOT}})$ that produces a random OLE correlation from $r = O(\lambda)$ random OT correlations in Fig. 6 below. Note that the superscript here denotes the choice, not power.

Lemma 5.3. *Let $\lambda \in \mathbb{N}$. Assuming random OT correlations over $\mathbb{F}_2$, there exists a two-round, semi-honest protocol Π_{fromOT} that outputs a single OLE correlation over $\mathbb{F}_{2^\lambda}$ and makes use of $O(\lambda)$ number of OT correlations.*

The communication complexity of Π_{fromOT} is $r + 2r + \lambda = O(\lambda)$. The constant in the big-O is roughly $3 \times 5.12 + 1 = 16.36$ as $\lambda \to \infty$.

For correctness of protocol Π_{fromOT}, first, by the definition of MFE, we have $\rho(u_1 v_1, \ldots, u_r v_r) = a_r x_r$. Next, consider $o_i = e_i^{C_i} \oplus R_i$. If $C_i = 0$, then we have

$$o_i = e_i^{C_i} \oplus R_i = t_i \oplus (u_i \cdot d_i) \oplus S_i^0 \oplus S_i^0 = t_i \oplus (u_i \cdot (v_i \oplus 0)) = t_i \oplus (u_i v_i).$$

Else $C_i = 1$, and we have

$$o_i = e_i^{C_i} \oplus R_i = t_i \oplus (u_i \cdot (1 \oplus d_i)) \oplus S_i^1 \oplus S_i^1 = t_i \oplus (u_i \cdot (1 \oplus v_i \oplus 1)) = t_i \oplus (u_i v_i).$$

Thus, in both cases $o_i = t_i \oplus (u_i v_i)$. In the end, we consider

$$\begin{aligned} y_r &= R + Y \\ &= R + T + b_r \\ &= \rho(o_1, \ldots, o_r) + \rho(t_1, \ldots, t_r) + b_r \\ &= \rho(o_1 \oplus t_1, \ldots, o_r \oplus t_r) + b_r \\ &= \rho(u_1 v_1, \ldots, u_r v_r) + b_r \\ &= a_r x_r + b_r, \end{aligned}$$

which explains the correctness of the protocol.

The semi-honest security of Π_{fromOT} is information-theoretic. In round 1 message sent by the receiver, each d_i is masked by the random choice bit C_i. Thus, π_1 is identically distributed as the uniform distribution over $\{0,1\}^r$. In round 2 message sent by the sender, each e_i^0 (resp. e_i^1) is masked by the random bit S_i^0 (resp. S_i^1), and Y is masked by b_r. Thus, π_2 is identically distributed as the uniform distribution over $\{0,1\}^{2r+\lambda}$.

We can combine $\Pi_{\mathsf{singleOLE}}$ with Π_{fromOT} to obtain a protocol that computes an OLE from random bit-OT correlations. Denote the combined protocol by $\Pi_{\mathsf{oleOT}} := (\Pi_{\mathsf{oleOT},1}, \Pi_{\mathsf{oleOT},2}, \mathsf{out}_{\mathsf{oleOT}})$ and we describe the protocol in Fig. 7.

To represent the correlated randomness r_{rec} and r_{sen}, it takes $O(\lambda)$ bits. Recall that the communication complexity of Π_{fromOT} is $O(\lambda)$. Π_{oleOT} also sends field elements $x', c, a' \in \mathbb{F}_{2^\lambda}$ in addition to π_1', π_2' in Π_{fromOT}. Therefore, the communication complexity of Π_{oleOT} is $O(\lambda)$.

Using the single OLE protocol Π_{oleOT} above, we construct an alternative inner protocol $\Pi^*_{\mathsf{OLE}} = (\Pi_{\mathsf{OLE},1}, \Pi_{\mathsf{OLE},2}, \mathsf{out}_{\Pi_{\mathsf{OLE}}})$ in Fig. 8, which is based on random bit-OT correlations.

Theorem 5.4. *Let $\lambda \in \mathbb{N}$. Assuming random bit-OT correlations, there exists a two-round, semi-honest protocol Π^*_{OLE} that computes $k \in \mathbb{N}$ instances of Oblivious Linear Evaluations (OLE) over $\mathbb{F}_{2^\lambda}$ in parallel, and the communication complexity is $O(k\lambda)$.*

To represent the correlated randomness $\mathbf{r}_{\mathsf{rec}}$ and $\mathbf{r}_{\mathsf{sen}}$, it takes $O(k\lambda)$ bits as each $r'_{\mathsf{rec},i}, r'_{\mathsf{sen},i}$ takes $O(\lambda)$ bits and each $x_{r,i}, a_{r,i}, b_{r,i}$ takes $O(\lambda)$ bits for $i \in [k]$.

In the outer protocol, we take $k = N' = N_{\mathsf{mono}}/\ell = O(|C|/\ell)$. Note that each of the m servers in the outer protocol runs an independent instance of Π^*_{OLE}. Taking m to be a constant multiple of ℓ, the total communication complexity is $O(|C|\lambda)$.

$\underline{\Pi_{\mathsf{oleOT}}}$:

- **Inputs:** The receiver has input $x \in \mathbb{F}$, and the sender has input $a, b \in \mathbb{F}$.
- **Setup:**
  - Fix a MFE $(\rho : \mathbb{F}_2^r \to \mathbb{F}_{2^\lambda},\ \mu : \mathbb{F}_{2^\lambda} \to \mathbb{F}_2^r)$ such that for any $a, x \in \mathbb{F}_{2^\lambda}$, we have $a \cdot x = \rho\big(u_1 v_1, \ldots, u_r v_r\big)$ where $u_i = \mu_i(a)$, $v_i = \mu_i(x)$ are the output of $\mu(a)$ (resp. $\mu(x)$) projected to the i-th bit.
  - Assume $r = O(\lambda)$ independent random bit-OT correlations are preprocessed: for each $i \in [r]$ the sender holds two bits $(S_i^0, S_i^1) \in \{0,1\}^2$ and the receiver holds $\big(C_i,\ R_i := S_i^{C_i}\big)$ with $C_i \leftarrow \{0,1\}$. Let $r_{\mathsf{rec}} := ((C_1, R_1), \ldots, (C_r, R_r))$ be the receiver's correlated randomness, and let $r_{\mathsf{sen}} := ((S_1^0, S_1^1), \ldots, (S_r^0, S_r^1))$ be the sender's correlated randomness.
- **Local Sampling:** The receiver samples $x_r \leftarrow \mathbb{F}_{2^\lambda}$. The sender samples $a_r, b_r \leftarrow \mathbb{F}_{2^\lambda}$.
- **Round 1:** The receiver runs $\underline{\Pi_{\mathsf{oleOT},1}(x, x_r, r_{\mathsf{rec}})}$ defined as follows:
  1. It runs $\pi_1' = \Pi_{\mathsf{fromOT},1}(x_r, r_{\mathsf{rec}}) \in \{0,1\}^r$.
  2. It computes $x' = x - x_r \in \mathbb{F}_{2^\lambda}$.
  3. It sends $\pi_1 = (\pi_1', x')$ to the sender.
- **Round 2:** The sender runs $\underline{\Pi_{\mathsf{oleOT},2}(\pi_1, a, b, a_r, b_r, r_{\mathsf{sen}})}$ defined as follows:
  1. It runs $\pi_2' = \Pi_{\mathsf{fromOT},2}(\pi_1', a_r, b_r, r_{\mathsf{sen}}) \in (\{0,1\}^{2r} \times \mathbb{F}_{2^\lambda})$.
  2. It computes $c = ax' + b - b_r$ and $a' = a - a_r$.
  3. It sends $\pi_2 = (\pi_2', c, a')$ to the receiver.
- **Output:** The receiver runs $\underline{\mathsf{out}_{\mathsf{oleOT}}(\pi_2, x_r, r_{\mathsf{rec}})}$ defined as follows:
  1. It computes $y_r = \mathsf{out}_{\mathsf{fromOT}}(\pi_2', r_{\mathsf{rec}})$.
  2. It computes $y = y_r + c + a' x_r$ and outputs y.

Figure 7: Performing a single OLE from OT correlations

The Inner Protocol $\underline{\Pi^*_{\mathsf{OLE}} = (\Pi_{\mathsf{OLE},1}, \Pi_{\mathsf{OLE},2}, \mathsf{out}_{\mathsf{OLE}})}$:

- **Inputs:** Receiver has input $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{F}_{2^\lambda}^k$, and sender has input $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{F}_{2^\lambda}^k$ and $\mathbf{b} = (b_1, \ldots, b_k) \in \mathbb{F}_{2^\lambda}^k$.
- **Setup:**
  - Fix a MFE $(\rho : \mathbb{F}_2^r \to \mathbb{F}_{2^\lambda},\ \mu : \mathbb{F}_{2^\lambda} \to \mathbb{F}_2^r)$ such that for any $a, x \in \mathbb{F}_{2^\lambda}$, we have $a \cdot x = \rho\big(u_1 v_1, \ldots, u_r v_r\big)$ where $u_i = \mu_i(a)$ (resp. $v_i = \mu_i(x)$) is the output of $\mu(a)$ (resp. $\mu(x)$) projected to the i-th bit.
  - Assume $kr = O(k\lambda)$ independent random bit-OT correlations are preprocessed: for each $i \in [k]$ and $j \in [r]$, the sender holds $(S^0_{i,j}, S^1_{i,j}) \in \{0,1\}^2$ and the receiver holds $\big(C_{i,j},\ R_{i,j} := S^{C_{i,j}}_{i,j}\big)$ with $C_{i,j} \leftarrow \{0,1\}$. For $i \in [k]$, let $r'_{\mathsf{rec,i}} := ((C_{i,1}, R_{i,1}), \ldots, (C_{i,r}, R_{i,r}))$ and let $r'_{\mathsf{sen,i}} := ((S^0_{i,1}, S^1_{i,1}), \ldots, (S^0_{i,r}, S^1_{i,r}))$.
- **Local Sampling:**
  - The receiver samples $\mathbf{x}_r := (x_{r,1}, \ldots, x_{r,k}) \leftarrow \mathbb{F}_{2^\lambda}^k$. Let $\mathbf{r}_{\mathsf{rec}} := (r_{\mathsf{rec},1}, \ldots, r_{\mathsf{rec,k}})$ be the receiver's correlated randomness, where $r_{\mathsf{rec,i}} := (x_{r,i}, r'_{\mathsf{rec,i}})$ for $i \in [k]$.
  - The sender samples $\mathbf{a}_r := (a_{r,1}, \ldots, a_{r,k}) \leftarrow \mathbb{F}_{2^\lambda}^k$ and $\mathbf{b}_r := (b_{r,1}, \ldots, b_{r,k}) \leftarrow \mathbb{F}_{2^\lambda}^k$. Let $\mathbf{r}_{\mathsf{sen}} := (r_{\mathsf{sen},1}, \ldots, r_{\mathsf{sen,k}})$ be the sender's correlated randomness, where $r_{\mathsf{sen,i}} := ((a_{r,i}, b_{r,i}), r'_{\mathsf{sen,i}})$ for $i \in [k]$.
- **Round 1:** The receiver runs $\underline{\Pi_{\mathsf{OLE},1}(\mathbf{x}, \mathbf{r}_{\mathsf{rec}})}$ defined as follows:
  1. For $i \in [k]$, it runs $\pi_{1,i} = \Pi_{\mathsf{oleOT},1}(x_i, x_{r,i}, r'_{\mathsf{rec,i}})$.
  2. It sends $\pi_1 = (\pi_{1,1}, \ldots, \pi_{1,k})$ to the sender.
- **Round 2:** The sender runs $\underline{\Pi_{\mathsf{OLE},2}(\pi_1, \mathbf{a}, \mathbf{b}, \mathbf{r}_{\mathsf{sen}})}$ defined as follows:
  1. For $i \in [k]$, it runs $\pi_{2,i} = \Pi_{\mathsf{oleOT},2}(\pi_{1,i}, a_i, b_i, a_{r,i}, b_{r,i}, r'_{\mathsf{sen,i}})$.
  2. It sends $\pi_2 = (\pi_{2,1}, \ldots, \pi_{2,k})$ to the receiver.
- **Output:** The receiver runs $\underline{\mathsf{out}_{\mathsf{OLE}}(\pi_2, \mathbf{r}_{\mathsf{rec}})}$ defined as follows:
  1. For $i \in [k]$, it runs $y_i = \mathsf{out}_{\mathsf{oleOT}}(\pi_{2,i}, x_{r,i}, r'_{\mathsf{rec,i}})$.
  2. It outputs $\mathbf{y} = (y_1, \ldots, y_k)$.

Figure 8: Semi-honest Inner Protocol from Random OT Correlations

6 Proofs of Reed-Solomon Proximity and Constraints

The security of the outer protocol holds only against the defined admissible adversary. In this section, we progressively remove the constraints imposed on the admissible adversary by replacing them with non-interactive zero-knowledge arguments in the random oracle model. Specifically, we present three argument systems addressing consistency constraints, linear constraints, and Boolean constraints, respectively.

Note that both clients need to prove to the other client that their secret sharing is honest. The sender proves for the shares of $\overline{p}^0, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$ to the receiver, and the receiver proves for the shares of $\overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ to the sender. We remark that the shares of the vector $\overline{p}^x$ (resp. $\overline{p}^y$) are deterministically computed using shares of $\overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ (resp. $\overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$) and will be checked by the compiler later.

Notations. Let $||$ denote the concatenation operation. To simplify the notation, we denote the prover's input (the shares of size-ℓ vectors it sends to the m servers) by an $n \times m$ matrix U where row $U_i := (u_{i,1}, \ldots, u_{i,m})$. For each $j \in [m]$, the column $U[j]$ stacks shares sent to server j vertically together. When the receiver client is the prover, column $U[j] = (\hat{x}_{j,1}||\hat{x}_{j,2}||\hat{x}_{j,3})$ (vertically stacked) and the number of rows $n = 3N'$. When the sender client is the prover, it needs to use two separate matrices. The first matrix is for the shares of the vectors $\overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$. In this case, column $U[j] = (\hat{y}_{j,1}||\hat{y}_{j,2}||\hat{y}_{j,3})$ (vertically stacked) and the number of rows $n = 3N'$. The second matrix is for the additive secret shares of zeros $\overline{p}^0$, in which case $U[j] := \hat{0}_j$ and the number of rows $n = N'$. We need two matrices because the degree of the polynomial underlying the packed Shamir secret sharing for $\overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$ and $\overline{p}^0$ are different. Let $(\mathsf{Com}, \mathsf{Open})$ be a non-interactive, straight-line extractable commitment scheme in the random oracle model [Pas03], where the commitment is of size $O(\lambda)$. With a polynomial number of queries to the random oracle, the commitments are statistically binding, computationally hiding, and straight-line extractable. We commit matrix U as follows. For each $j \in [m]$, we commit column $U[j]$ by computing $\mathsf{com}_j := \mathsf{Com}(U[j], s_j)$ where $s_j \leftarrow \{0,1\}^\lambda$ is a uniform random salt. For simplicity and readability, in this section, we hide the random salt parameter and write $\mathsf{Com}(U[j])$ instead. Furthermore, we define $\overline{\mathsf{com}} := (\mathsf{com}_1, \ldots, \mathsf{com}_m)$ as the commitment for matrix U. The random oracle is modeled by hash functions $\mathcal{H} := (H_{\mathsf{com}}, H_T, H_r)$, where H_{com} is used in the commitment scheme $(\mathsf{Com}, \mathsf{Open})$, $H_T : \{0,1\}^* \to S_{m,|T|}$ outputs a uniform random subset $T \subseteq [m]$ and $|T|$ is a constant fraction of m, and $H_r : \{0,1\}^* \to \{0,1\}^*$ generates a random binary string that can be interpreted as field elements of $\mathbb{F}$.

6.1 The Proof System for Consistency Requirements

Recall packed Shamir secret sharing of a vector $v = (v_1, \ldots, v_\ell)$. Fix m distinct points $\overline{\alpha} = (\alpha_1, \ldots, \alpha_m) \in \mathbb{F}^m$ and ℓ distinct points $\overline{\beta} = (\beta_1, \ldots, \beta_\ell) \in \mathbb{F}^\ell$ and $\overline{\alpha}$ and $\overline{\beta}$ are disjoint. We sample a random polynomial p such that $(p(\beta_1), \ldots, p(\beta_\ell)) = (v_1, \ldots, v_\ell)$. The sharing is then $p(\alpha_1), \ldots, p(\alpha_m)$. Since the shares are evaluations of a polynomial, we can view each sharing (i.e. each row of matrix U) as a Reed-Solomon codeword. Let $k = t + \ell$. Let $\mathsf{RS}_{\overline{\alpha},\overline{\beta}}[\mathbb{F}, k, m]$ denote the Reed-Solomon code of dimension $k+1$ and block length m over finite field $\mathbb{F}$, where the actual message is evaluated on $\overline{\beta} \in \mathbb{F}^\ell$. In our case, $\overline{\alpha}, \overline{\beta}$ are pre-determined before the execution of the outer protocol. Thus, to simplify the notation, we write $\mathsf{RS}[\mathbb{F}, k, m]$ when the context is clear. For any vector $v \in \mathbb{F}^\ell$, to compute the RS codeword, we sample a random polynomial p of degree k conditioned on $p(\beta_i) = v_i$ for all $i \in [\ell]$, then the codeword is $(p(\alpha_1), \ldots, p(\alpha_m)) \in \mathsf{RS}[\mathbb{F}, k, m]$. If

the secrets are honestly shared, then for each $i \in [n]$, $U_i := (u_{i,1}, \ldots, u_{i,m})$ should be an exact RS codeword. For $i \in [n]$, let p_i be the degree-k polynomial used in the packed Shamir secret sharing.

Description of the Non-Interactive Argument System. In Fig. 9 below, we describe the non-interactive argument system, denoted by $(P, V)^{\mathcal{H}}_{\mathsf{rs}}$, to test Reed-Solomon proximity, which shows the shares are consistent with RS codewords in $\mathsf{RS}[\mathbb{F}, k, m]$. This is essentially a zero-knowledge version of the Ligero test [AHIV17] for interleaved Reed-Solomon codes, applied with the Fiat-Shamir transformation using the random oracle.

The input of the prover consists of the matrix U and the commitments of U's columns $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$, and as witnesses the polynomials $p_1, \ldots, p_n$ used in the packed secret sharing of the rows $U_1, \ldots, U_n$ respectively. The input of the verifier is the commitments $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$. We remark that the prover also obtains the associated random salts used in the commitments implicitly. When the prover opens a commitment to the verifier, it also provides the random salt as part of the opening. Both the prover and the verifier have access to the random oracle $\mathcal{H} = (H_{\mathsf{com}}, H_T, H_r)$, where H_{com} is used to create the commitments $\overline{\mathsf{com}}$. The size of the random subset T should be a constant fraction of m, and it can be tweaked as a parameter according to the desired concrete security level in practice. In our final compiled protocol, we invoke $(P, V)^{\mathcal{H}}_{\mathsf{rs}}$ with $|T| = \lambda$.

Communication Complexity Consider the message π_{rs} sent by the prover. The commitment $\overline{\mathsf{com}}'$ takes $m \cdot O(\lambda)$ bits. The random salts s_r, s_T takes λ bits each. The result of linear combination $\overline{u}'$ takes $m \log |\mathbb{F}| = m\lambda$ bits. To represent the subset T, it takes $|T| \log m$ bits. The openings $\{\mathsf{open}'_j : j \in T\}$ take $|T| \cdot O(\lambda)$ bits. The openings $\{\mathsf{open}_j : j \in T\}$ take $|T| \cdot O(n\lambda)$ bits as they need to contain the whole column $U[j]$. Summing them up, the total communication is $O(m\lambda) + 2\lambda + m\lambda + |T| \log m + |T| \cdot O(\lambda) + |T| \cdot O(n\lambda)$, which is dominated by the $|T| \cdot O(n\lambda)$ term. Recall that $n \leq 3N'$ and $N' \cdot \ell = N_{\mathsf{mono}} = O(|C|)$. Therefore, the communication complexity of $(P, V)^{\mathcal{H}}_{\mathsf{rs}}$ is $|T| \cdot O(|C|\lambda/\ell)$. If we take $\ell = O(\lambda)$ and $|T| = \lambda$, then the communication complexity becomes $O(|C|\lambda)$.

Completeness. For completeness, it follows directly from the description. Note that $\overline{u}'$ is a linear combination of RS codewords $U_0, U_1, \ldots, U_n$. By linearity of Reed-Solomon code, $\overline{u}'$ is in $\mathsf{RS}[\mathbb{F}, k, m]$ with probability 1.

Soundness. For soundness, note that we share n vectors, and each sharing U_i can be viewed as a (possibly corrupted) RS codeword. Let t be the number of corrupted servers defined in the outer protocol, which is also the number of corrupted columns in U. If the union of all corrupted positions over all n received codewords has size at most t, which means at most t columns are corrupted, then by the error-correcting property of the underlying secret sharing scheme, the secrets can still be recovered.

Before proving the soundness, we introduce some notations. Let U' be a $(n+1) \times m$ matrix with rows $U_0, U_1, \ldots, U_n$. Let $U_i := (u_{i,1}, \ldots, u_{i,m})$ for $i \in \{0, \ldots, n\}$. Let L^{n+1} be a $(n+1) \times m$ matrix, where each row $L_i \in \mathsf{RS}[\mathbb{F}, k, m]$ is a closest Reed–Solomon codeword in Hamming distance to U_i. Let $d(U_i, L_i)$ denote the Hamming distance between U_i and L_i. Furthermore, for any string $s \in \mathbb{F}^m$, we use $d(s, \mathsf{RS})$ to denote the Hamming distance between s and its closest codeword in $\mathsf{RS}[\mathbb{F}, k, m]$. Let $\Delta(U_i, L_i) := \{j \in [m] : U_{i,j} \neq L_{i,j}\}$ denote the set of indices where U_i disagrees

$\underline{(P,V)^{\mathcal{H}}_{\mathsf{rs}}}$:

- **Input**: The purported shares represented by matrix U and the commitments $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$. The witnesses $(p_1, \ldots, p_n)$ which are polynomials of degree k. Note that $\overline{\mathsf{com}}$ is the common input of both the prover and the verifier.
- **Prover** $P^{\mathcal{H}}_{\mathsf{rs}}(\overline{\mathsf{com}}, U, (p_1, \ldots, p_n))$:
  1. (Insert random row) It samples a random degree-k polynomial p_0. Then it computes $U_0 := (p_0(\alpha_1), \ldots, p_0(\alpha_m)) = (u_{0,1}, \ldots, u_{0,m})$.
  2. For each $j \in [m]$, it computes $\mathsf{com}'_j := \mathsf{Com}(\mathsf{com}_j || u_{0,j})$. Let $\overline{\mathsf{com}}' := (\mathsf{com}'_1, \ldots, \mathsf{com}'_m)$.
  3. It samples a random salt $s_r \leftarrow \{0,1\}^\lambda$. Then, it uses the random oracle to generate a random column vector $r := H_r(\overline{\mathsf{com}}, \overline{\mathsf{com}}', s_r)$, where $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$.
  4. (Folding) It computes $\overline{u}' := r^\intercal U + U_0$, and denote the elements of $\overline{u}'$ as $(u'_1, \ldots, u'_m) \in \mathbb{F}^m$.
  5. It samples a random salt $s_T \leftarrow \{0,1\}^\lambda$. Then it uses the random oracle to obtain a random subset $T := H_T(\overline{\mathsf{com}}, \overline{\mathsf{com}}', \overline{u}', s_T) \subseteq [m]$ and $|T| \le t$.
  6. For each $j \in T$, it computes the openings $\mathsf{open}'_j := \mathsf{Open}(\mathsf{com}'_j)$ and $\mathsf{open}_j := \mathsf{Open}(\mathsf{com}_j)$. Specifically, open'_j should contain $(\mathsf{com}_j || u_{0,j})$ and open_j should contain $U[j]$.
  7. It outputs $\pi_{\mathsf{rs}} := \big(\overline{\mathsf{com}}', s_r, s_T, \overline{u}', T, \{\mathsf{open}_j : j \in T\}, \{\mathsf{open}'_j : j \in T\}\big)$.
- **Verifier** $V^{\mathcal{H}}_{\mathsf{rs}}(\overline{\mathsf{com}}, \pi_{\mathsf{rs}})$:
  1. It calls $H_T(\overline{\mathsf{com}}, \overline{\mathsf{com}}', \overline{u}', s_T)$ and checks if it is equal to T sent by the prover. If so, it checks for each $j \in T$ if the openings are successful, i.e. open'_j should successfully open $(\mathsf{com}_j || u_{0,j})$ and open_j should successfully open $U[j]$.
  2. It calls the random oracle $H_r(\overline{\mathsf{com}}, \overline{\mathsf{com}}', s_r)$ to obtain $r = (r_1, \ldots, r_n)$.
  3. It does the following:
     (a) For each $j \in T$, it computes $r^\intercal U[j] + u_{0,j}$ and checks if it is equal to u'_j sent by the prover.
     (b) It checks if $\overline{u}' \in \mathsf{RS}[\mathbb{F}, k, m]$ (i.e. is an exact RS codeword).
  4. It accepts if and only if all the checks above pass.

Figure 9: Description of $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ for Consistency Requirements

with L_i. Let $h(U', L^{n+1}) := \left|\cup_{i \in \{0,\ldots,n\}} \Delta(U_i, L_i)\right|$ denote the size of the union of indices where U_i disagrees with L_i, over all $i \in \{0, 1, \ldots, n\}$.

To prove soundness, the intuition is that if the rows of U' are not all exact RS codewords, then a linear combination of the rows of U' is unlikely to be close to a RS codeword. This is formally studied as the proximity gap theorem. We state a proximity gap theorem by Ben-Sasson et al. [BCI+20] below. Note that we only take t up to the unique decoding bound and we take $t = \Theta(m)$ and $m = \Theta(\lambda)$ and $|\mathbb{F}| = 2^\lambda$. Applying this theorem, we prove the soundness of $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$, which is negligible in λ.

Theorem 6.1 (Rephrase of Theorem 1.6 from [BCI+20])**.** *For any $n+1$ strings $U_0, U_1, \ldots, U_n \in \mathbb{F}^m$, let $S := U_0 + \mathrm{span}\{U_1, \ldots, U_n\} \subset \mathbb{F}^m$ be an affine subspace. If $t \in (0, (m-k)/2)$, and*

$$\Pr_{s \in S}[d(s, \mathsf{RS}) \le t] > m/|\mathbb{F}|,$$

then there exists a subset of positions $\mathcal{D}' \subseteq [m]$ and $n+1$ exact Reed-Solomon codewords $L_0, L_1, \ldots, L_n \in \mathsf{RS}[\mathbb{F}, k, m]$ satisfying $|\mathcal{D}'| \ge m - t$ and for all $i \in \{0, 1, \ldots, n\}$, string U_i and codeword L_i agree on all of $\mathcal{D}'$.

Recall the commitment scheme $(\mathsf{Com}, \mathsf{Open})$ from random oracle is straight-line extractable. Let Ext denote the polynomial time extractor of the commitment scheme.

Lemma 6.2 (Soundness of $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$)**.** *Let $\mathbb{F}$ be a finite field with size 2^λ. Fix some integer $t \in (0, (m-k)/2)$ where $t = \Theta(m)$ and $m = \Theta(\lambda)$. Take $|T| = \Theta(t)$. Let ϵ be the error probability of the extractor of commitment scheme $(\mathsf{Com}, \mathsf{Open})$, where ϵ is negligible in λ.*

Assume the cheating prover can make at most $\mathsf{poly}(\lambda)$ queries to the random oracle $\mathcal{H}$. Let $U' \in \mathbb{F}^{(n+1)\times m}$ be the matrix extracted by the extractor Ext, and the rows of U' are $U_0^*, U_1^*, \ldots, U_n^*$. Let $L^{n+1} \in \mathbb{F}^{(n+1)\times m}$ be the matrix formed by rows $L_0, L_1, \ldots, L_n \in \mathbb{F}^m$, where for $i \in \{0,1,\ldots,n\}$, row $L_i \in \mathsf{RS}[\mathbb{F}, k, m]$ is the RS codeword closest to U_i^*. Recall $h(U', L^{n+1})$ is defined as the union of corrupted columns of U'. If $h(U', L^{n+1}) > t$, then the verifier accepts with negligible probability in λ.

Proof. Observe the queries that the cheating prover makes to the random oracle. For each query, consider the following scenario below.

With probability ϵ, the extractor may extract a different matrix from the one actually used by the cheating prover. Therefore, the probability that the verifier accepts is upper bounded by ϵ. Let $U^* \in \mathbb{F}^{n\times m}$ be the matrix formed by rows $U_1^*, \ldots, U_n^*$. In other words, U' is U_0^* juxtaposed on top of U^*. Let $\overline{u}^* = r^\intercal U^* + U_0^*$. Conditioned on successful extraction, consider the following two cases.

Case 1. The number of corrupted positions $d(\overline{u}^*, \mathsf{RS}) > t$. Note that if the cheating prover wants the verifier to accept, it must send an exact RS codeword $\overline{u}' \neq \overline{u}^*$ and thus it cannot honestly execute the computation $r^\intercal U^* + U_0^*$. Let J be the set of indices that $\overline{u}'$ disagrees with $\overline{u}^*$. If $J \cap T \neq \emptyset$, the verifier will catch the cheating prover. The verifier accepts when $J \cap T = \emptyset$. Since $d(\overline{u}^*, \mathsf{RS}) > t$, we have $|J| > t$. Since T is chosen uniformly at random by the random oracle H_T, the probability that $J \cap T = \emptyset$ is upper bounded by $(1 - t/m)^{|T|}$. Note that we take $|T| = \Theta(t)$, and t to be a constant fraction of m and $m = \Theta(\lambda)$.

Case 2. The number of corrupted positions $d(\overline{u}^*, \mathsf{RS}) \leq t$. In this case, we upper bound the event that the verifier accepts by $\Pr[d(\overline{u}^*, \mathsf{RS}) \leq t]$. Note that vector r is output of the random oracle H_r, and $\overline{u}^* = r^\intercal U^* + U_0^*$. Therefore, $\overline{u}^*$ is a uniform random element in the affine subspace $S = U_0^* + \mathrm{span}\{U_1^*, \ldots, U_n^*\} \subset \mathbb{F}^m$. Recall we assume $h(U', L^{n+1}) > t$ in the lemma statement, i.e., the number of corrupted columns in U' is more than t. In other words, the number of uncorrupted columns of U' is strictly less than $m - t$. Therefore, there does not exist a subset of positions $\mathcal{D}' \subseteq [m]$ of size $\geq m - t$ where U_i^* agrees with L_i on all of $\mathcal{D}'$ for all $i \in \{0,1,\ldots,n\}$. Applying the contrapositive of Theorem 6.1, we have

$$\Pr_{s\in S}[d(s, \mathsf{RS}) \leq t] \leq m/|\mathbb{F}| \tag{11}$$

for $t \in (0, (m-k)/2)$. Recall that $\overline{u}^*$ is a uniform random vector in the affine subspace S. By equation (11), we have that $\Pr[d(\overline{u}^*, \mathsf{RS}) \leq t] \leq m/|\mathbb{F}|$. Note that $m = \Theta(\lambda)$ and $|\mathbb{F}| = 2^\lambda$.

Finally, we union bound by the (polynomial number of) queries sent by the cheating prover to the random oracle, the probability that the verifier accepts is then upper bounded by $\mathsf{poly}(\lambda) \cdot \big((1 - t/m)^{|T|} + m/|\mathbb{F}| + \epsilon\big)$, which is negligible in λ. □

Zero-knowledge. The zero-knowledge of $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ means that the verifier should not be able to learn about the $n \times m$ matrix U. Consider a different matrix U^*, which is a RS encoding of a different set of n vectors of length ℓ. The verifier's view should be indistinguishable between the prover using U and using U^*. Note that the argument system is non-interactive, so we only need to show honest-verifier zero-knowledge. We give the zero-knowledge lemma statement below.

Lemma 6.3 (Zero-knowledge of $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$)**.** *Assume the verifier can make at most* $\mathsf{poly}(\lambda)$ *queries to the random oracle* $\mathcal{H}$*. The non-interactive argument system* $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ *is zero-knowledge.*

Proof. Consider a simulator S that works as follows:

1. It observes the queries the verifier sends to the random oracle.
2. It samples a random RS codeword $\overline{u}^* \leftarrow \mathsf{RS}[\mathbb{F}, k, m]$ where $\overline{u}^* := (\overline{u}^*_1, \ldots, \overline{u}^*_m)$.
3. It samples a random subset $T^* \subseteq [m]$ where $|T^*| = |T|$. It samples matrix U^* as follows: For $j \in T^*$, it samples random columns $U^*[j] \leftarrow \mathbb{F}^n$. For $j \notin T^*$, columns $U^*[j]$ are all-zero columns.
4. It samples random coefficients $r^* \leftarrow \mathbb{F}^n$. For $j \in T^*$, it computes $u^*_{0,j} = \overline{u}^*_j - (r^*)^\intercal U^*[j]$. For $j \notin T^*$, it sets $u^*_{0,j} = 0$.
5. It commits $\mathsf{com}^*_j = \mathsf{Com}(U^*[j])$ and $\mathsf{com}^{*'}_j = \mathsf{Com}(\mathsf{com}^*_j || u^*_{0,j})$ for all $j \in [m]$. Let $\overline{\mathsf{com}}^* := (\mathsf{com}^*_1, \ldots, \mathsf{com}^*_m)$ and $\overline{\mathsf{com}^*}' := (\mathsf{com}^{*'}_1, \ldots, \mathsf{com}^{*'}_m)$.
6. It samples a random salt $s_r \leftarrow \{0,1\}^\lambda$ and it programs the random oracle $H_r(\overline{\mathsf{com}}^*, \overline{\mathsf{com}^*}', s_r)$ to return r^*. If the verifier has already made the same query to the random oracle, then abort.
7. It samples a random salt $s_T \leftarrow \{0,1\}^\lambda$ and programs the random oracle $H_T(\overline{\mathsf{com}}^*, \overline{\mathsf{com}^*}', \overline{u}^*, s_T)$ to return T^*. If the same query has already been made by the verifier, abort.
8. It computes the openings $\mathsf{open}^{*\prime}_j$ and open^*_j for all $j \in T^*$. Specifically, $\mathsf{open}^{*\prime}_j$ should contain $(\mathsf{com}^*_j || u^*_{0,j})$ and open^*_j should contain $U^*[j]$.
9. It sends $\pi^*_{\mathsf{rs}} := \left(\overline{\mathsf{com}^*}', s_r, s_T, \overline{u}^*, T^*, \{\mathsf{open}^*{}_j : j \in T^*\}, \{\mathsf{open}^{*\prime}{}_j : j \in T^*\}\right)$.

Consider the following hybrids:

1. Hyb_0: This corresponds to the real execution where the verifier interacts with the prover in $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$.
2. Hyb_1: In this hybrid, when the verifier queries the random oracle H_T, we return T^* sampled by S. If the verifier has already made the same query, we abort. Since s_T sampled by the simulator has λ bits, the probability that it aborts is at most $2^{-\lambda}$. Since T^* is sampled uniformly at random, and by property of random oracle, T^* is identically distributed as T. Note that $|T| = |T^*| \leq t$. By the t-privacy in the packed secret sharing scheme, we have $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.
3. Hyb_2: In this hybrid, when the verifier queries the random oracle H_r, we return r^* sampled by S. Since r^* is sampled uniformly at random, and by property of the random oracle, r^* is identically distributed as r, and thus $\overline{u}' = r^\intercal U + U_0 \equiv (r^*)^\intercal U + U_0$. If the verifier has already made the same query, we abort. Since s_r sampled by S has λ bits, the probability it aborts is at most $2^{-\lambda}$. Thus, we have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

4. $\underline{\mathsf{Hyb}_3}$: For $j \in T^*$, we replace column $U[j]$ with random column $U^*[j] \leftarrow \mathbb{F}^n$. We also sample $\overline{u^*} \leftarrow \mathsf{RS}[\mathbb{F}, k, m]$ to be a uniform random RS codeword to replace $\overline{u}'$. For $j \in T^*$, we compute $u^*_{0,j} = \overline{u}^*_j - (r^*)^\intercal U^*[j]$. Note that $u^*_{0,j}$ is also uniform random. Then for $j \in T^*$, we commit $\mathsf{com}^* = \mathsf{Com}(U^*[j])$ and $\mathsf{com}^{*'} = \mathsf{Com}(\mathsf{com}^* || u^*_{0,j})$.

   Note that $|T^*| \leq t$. By the t-privacy of the packed secret sharing scheme, the opened values $U[j]$ and $u_{0,j}$ are identically distributed as the random columns $U^*[j]$ and $u^*_{0,j}$ for $j \in T^*$. By the hiding property of the commitment scheme, the commitments in this hybrid and previous hybrid on columns $j \in T^*$ are computationally indistinguishable. Therefore, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_3$.

5. $\underline{\mathsf{Hyb}_4}$: For columns $U^*[j]$ and $u^*_{0,j}$ where $j \notin T^*$, we replace them as zeros. For $j \notin T^*$, we commit $\mathsf{com}^* = \mathsf{Com}(U^*[j])$ and $\mathsf{com}^{*'} = \mathsf{Com}(\mathsf{com}^* || u^*_{0,j})$. Note that for $j \notin T^*$, $U^*[j]$ and $u^*_{0,j}$ are not revealed to the verifier. By the hiding property of the commitment scheme, we have $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

Note that Hyb_4 corresponds to the simulator. Hence, the verifier's view in the real world is indistinguishable from its view in when interacting with S. □

6.2 The Proof System for Linear Constraints

Recall the linear constraints requirement of the admissible adversary, where the receiver (resp. sender) needs to show that the vectors they share satisfy linear constraints $\mathcal{R}_x$ (resp. $\mathcal{R}_y$) defined as a system of linear equations $Av = b$. To remove the linear constraints requirement of the admissible adversary, we describe a non-interactive argument system $(P, V)^{\mathcal{H}}_{\mathsf{lin}}$, which is essentially the zero-knowledge version of Ligero test for linear constraints [AHIV17], combined with a Fiat-Shamir transformation in the random oracle model.

To simplify the notation, let $v_1, \ldots, v_n \in \mathbb{F}^\ell$ be the vectors to be packed secret shared by the prover. Specifically, when the receiver is the prover, we set vectors $v_1 || \ldots || v_n := \overline{p}^x_1 || \overline{p}^x_2 || \overline{p}^x_3$; when the sender is the prover, we set $v_1 || \ldots || v_n := \overline{p}^y_1 || \overline{p}^y_2 || \overline{p}^y_3$ and separately for additive secret shares of zeros $v_1 || \ldots || v_n := \overline{p}^0$. Since the vectors here are by default column vectors, we abuse the notation $||$ to mean vertical stacking of column vectors. Let $p_1, \ldots, p_n$ be the degree-k polynomials used in the packed Shamir secret sharing of $v_1, \ldots, v_n$ respectively. Let $v \in \mathbb{F}^{n\ell}$ be the vector defined as $v := v_1 || \ldots || v_n$ and we describe the linear constraints $\mathcal{R}_x$ (resp. $\mathcal{R}_y$) as the linear system $Av = b$, where $A \in \mathbb{F}^{n\ell \times n\ell}$ is a matrix and $b := (b_{11}, \ldots, b_{1\ell}, \ldots, b_{n1}, \ldots, b_{n\ell}) \in \mathbb{F}^{n\ell}$ is a vector.

The input to the prover is a matrix U. Additionally, the prover and the verifier has common input $\overline{\mathsf{com}}$, which are commitments of the columns of U. The witness to the prover is the polynomials $p_1, \ldots, p_n$, which are polynomials used in the packed secret sharing of the rows of U. If the prover is honest, each row U_i is a Reed-Solomon encoding of v_i through polynomial p_i. Thus, we can first run $(P, V)^{\mathcal{H}}_{\mathsf{rs}}$ described in the previous section to ensure that U is t-close (disagreed columns are at most t) to n interleaved RS codewords in $\mathsf{RS}[\mathbb{F}, k, m]$. Assuming this, the prover wants to prove that the vectors recovered from the shares (represented by the matrix U) satisfy the linear constraints $Av = b$. We describe the non-interactive argument system $(P, V)^{\mathcal{H}}_{\mathsf{lin}}$ in Figure 10 below. This can be used to show that $\overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3$ (resp. $\overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$) satisfy linear constraints $\mathcal{R}_x$ (resp. $\mathcal{R}_y$). To prove for the additive secret shares of zeros, the argument system is similarly defined except that the degree of p_0 and $q(\cdot)$ become $3k + \ell - 1$ since we share $\overline{p}^0$ using a degree-$3k$ polynomial.

We remark that we need to additionally run $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ on the matrix U' to ensure it is t-close to $(n+1)$ interleaved RS codewords in L^{n+1}, where U' is defined as prover's matrix U and its additional row U_0 used for masking. Note that one can combine $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ and $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ into a single non-interactive argument system that ensures both the Reed-Solomon closeness and the linear constraints. The combined proof will have the same asymptotic communication complexity but better constants. We present $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ and $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ separately for better readability.

Communication Complexity Consider the message sent by the prover in $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$:

- $\overline{\mathsf{com}}'$: $O(m\lambda)$ bits.
- s_r, s_T: 2λ bits.
- $\bar{c}$: $(k+\ell)\log|\mathbb{F}| = (k+\ell)\lambda$ bits.
- T: $|T|\log m$ bits.
- $\{\mathsf{open}'_j : j \in T\}$: $|T| \cdot O(\lambda)$ bits.
- $\{\mathsf{open}_j : j \in T\}$: $|T| \cdot O(n\lambda)$ bits

The total communication is $O(m\lambda) + 2\lambda + (k+\ell)\lambda + |T|\log m + |T| \cdot O(\lambda) + |T| \cdot O(n\lambda)$, which is dominated by the $|T| \cdot O(n\lambda)$ term. Recall that $n \le 3N'$ and $N' \cdot \ell = N_{\mathsf{mono}} = O(|C|)$. Thus, the communication complexity is $|T| \cdot O(|C|\lambda/\ell)$. If we take $\ell = \lambda$, $|T| = \Theta(m)$, $m = 8\lambda$, then the communication complexity is $O(|C|\lambda)$.

Completeness. For correctness of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$, if $Av = b$, then $r^\intercal Av = r^\intercal b$. Note that the additionally sampled $p_0(\cdot)$ satisfies $\sum_{c=1}^{\ell} p_0(\beta_c) = 0$. The "folding step" of the prover computes $q(\cdot) = p_0(\cdot) + \sum_{i=1}^{n} r_i(\cdot) \cdot p_i(\cdot)$. Recall that for $i \in [n]$, we have $r_i(\beta_c) = r_{ic}$ for all $c \in [\ell]$. Therefore, we have $\sum_{c=1}^{\ell} q(\beta_c) = \sum_{c=1}^{\ell} p_0(\beta_c) + \sum_{i\in[n],c\in[\ell]} r_{ic} \cdot b_{ic} = \sum_{i\in[n],c\in[\ell]} r_{ic} \cdot b_{ic}$ since we sample p_0 conditioned on $\sum_{c=1}^{\ell} p_0(\beta_c) = 0$. This explains the correctness of the verification step.

Soundness. For soundness, observe that $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ is the Generalized-Affine-Test-Linear-Constraints-IRS test described in Ligero applied with the Fiat-Shamir transformation using the random oracle. Let $L = \mathsf{RS}[\mathbb{F}, k, m]$ be a Reed-Solomon code with distance $d = m - k$. We state Lemma B.7 from Ligero below.

Lemma 6.4 (Lemma B.7 from [AHIV17])**.** *Let t be a positive integer such that $t < d/2$. Suppose that a (badly formed) oracle U^* that is vertically juxtaposed with an arbitrary $U_0^* \in \mathbb{F}^m$ is t-close to a codeword $V \in L^{n+1}$, where V contains the codewords $U \in L^n$ and $U_0 \in L$ vertically juxtaposed, and U encodes $v \in \mathbb{F}^{n\ell}$ such that $Av \neq b$. Then, for any malicious P strategy, U^* is rejected by verifier except with at most $1/|\mathbb{F}| + ((t+k+\ell)/m)^{|T|}$ probability.*

The soundness of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ follows from Lemma 6.4. Note that the t-closeness in Ligero is defined as the maximum Hamming distance between any row and its closest RS codeword in $\mathsf{RS}[\mathbb{F}, k, m]$. While in our definition, the t-closeness is defined as the union of all corrupted columns. Let U' be the matrix obtained by vertically juxtaposing U_0^* above U^*. If U' is t-close to L^{n+1} according to our union-of-corrupted-columns distance, then it is also t-close according to Ligero's maximum-row Hamming distance. However, the converse need not hold. Our observation is that the same proof structure in Ligero also works for our case regardless of how t-closeness is defined. We give the soundness lemma below. Our proof follows the same structure as in [AHIV17].

$\underline{(P,V)^{\mathcal{H}}_{\mathsf{lin}}}$:

- **Input**: The purported shares represented as matrix U and their commitments $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$. The witnesses $(p_1, \ldots, p_n)$ which are polynomials of degree k. Note that $\overline{\mathsf{com}}, A, b$ are the common inputs to both the prover and the verifier.
- **Prover** $P^{\mathcal{H}}_{\mathsf{lin}}(\overline{\mathsf{com}}, U, (p_1, \ldots, p_n), A, b)$:
  1. It samples a random degree-$(k+\ell-1)$ polynomial p_0 under the constraint that $\sum_{c=1}^{\ell} p_0(\beta_c) = 0$. Let vector $v_0 := (p_0(\beta_1), \ldots, p_0(\beta_\ell))$ and let row $U_0 := (p_0(\alpha_1), \ldots, p_0(\alpha_m)) = (u_{0,1}, \ldots, u_{0,m})$. Then it computes $\overline{\mathsf{com}}' := (\mathsf{com}'_1, \ldots, \mathsf{com}'_m)$, where $\mathsf{com}'_j := \mathsf{Com}(\mathsf{com}_j || u_{0,j})$ for each $j \in [m]$.
  2. It samples a random salt $s_r \leftarrow \{0,1\}^\lambda$. Then, it uses the random oracle to obtain $r := H_r(\overline{\mathsf{com}}, \overline{\mathsf{com}}', s_r)$ where $r \in \mathbb{F}^{n\ell}$.
  3. It computes $r^\intercal A = (r_{11}, \ldots, r_{1\ell}, \ldots, r_{n1}, \ldots, r_{n\ell})$. For $i \in [n]$, let $r_i(\cdot)$ be the unique polynomial of degree $\ell - 1$ such that $r_i(\beta_c) = r_{ic}$ for $c \in [\ell]$.
  4. (Folding) It computes a degree-$(k+\ell-1)$ polynomial $q(\cdot) := p_0(\cdot) + \sum_{i=1}^{n} r_i(\cdot) \cdot p_i(\cdot)$. Denote the coefficients of $q(\cdot)$ by a size-$(k+\ell)$ vector $\overline{c} := (c_1, \ldots, c_{k+\ell})$.
  5. It samples a random salt $s_T \leftarrow \{0,1\}^\lambda$. Then it uses the random oracle to choose a subset $T := H_T(\overline{\mathsf{com}}, \overline{\mathsf{com}}', \overline{c}, s_T) \subseteq [m]$ where $|T| = \Theta(m)$.
  6. For each $j \in T$, it computes the openings $\mathsf{open}'_j := \mathsf{Open}(\mathsf{com}'_j)$ and $\mathsf{open}_j := \mathsf{Open}(\mathsf{com}_j)$. Specifically, open'_j should contain $(\mathsf{com}_j || u_{0,j})$ and open_j should contain $U[j]$.
  7. It outputs $\pi_{\mathsf{lin}} := \big(\overline{\mathsf{com}}', s_r, s_T, \overline{c}, T, \{\mathsf{open}'_j : j \in T\}, \{\mathsf{open}_j : j \in T\}\big)$.
- **Verifier** $V^{\mathcal{H}}_{\mathsf{lin}}(\overline{\mathsf{com}}, A, b, \pi_{\mathsf{lin}})$:
  1. It calls $H_T(\overline{\mathsf{com}}, \overline{\mathsf{com}}', \overline{c}, s_T)$ and checks if it is equal to T sent by the prover. Then, it checks for each $j \in T$ if the openings are successful, i.e. open'_j should successfully open $(\mathsf{com}_j || u_{0,j})$ and open_j should successfully open $U[j]$.
  2. It calls the random oracle $H_r(\overline{\mathsf{com}}, \overline{\mathsf{com}}', s_r)$ to obtain the vector r. Then it computes $r^\intercal A = (r_{11}, \ldots, r_{1\ell}, \ldots, r_{n1}, \ldots, r_{n\ell})$. For $i \in [n]$, let $r_i(\cdot)$ be the unique polynomial of degree $\ell - 1$ such that $r_i(\beta_c) = r_{ic}$ for $c \in [\ell]$.
  3. It recovers the polynomial $q(\cdot)$ from $\overline{c}$ and does the following:
     (a) For each $j \in T$, it computes $u_{0,j} + \sum_{i=1}^{n} r_i(\alpha_j) \cdot u_{i,j}$ and checks if it is equal to $q(\alpha_j)$ sent by the prover.
     (b) It checks if $\sum_{c=1}^{\ell} q(\beta_c) = \sum_{i \in [n], c \in [\ell]} r_{ic} \cdot b_{ic}$.
  4. It accepts if and only if all the checks above pass.

Figure 10: Description of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ for Linear Constraints

Lemma 6.5 (Soundness of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$)**.** *Let $\mathbb{F}$ be a finite field with size 2^λ. Fix some integer $t \in (0,(m-k)/2)$ where $t=\Theta(m)$ and $m=\Theta(\lambda)$. Take $|T|=\Theta(t)$. Let ϵ be the error probability of the extractor Ext of commitment scheme $(\mathsf{Com},\mathsf{Open})$, where $\epsilon=\mathsf{negl}(\lambda)$.*

Assume the cheating prover can make at most $\mathsf{poly}(\lambda)$ queries to the random oracle $\mathcal{H}$. Let $U' \in \mathbb{F}^{(n+1)\times m}$ be the matrix extracted by the extractor Ext, and the rows of U' are $U_0^*, U_1^*, \ldots, U_n^*$. Suppose U' is t-close to a codeword $V \in L^{n+1}$, where V contains the codeword $U_0 \in L$ vertically juxtaposed on top of n interleaved codewords $U \in L^n$, and each row $U_i \in \mathbb{F}^m$ encodes $v_i \in \mathbb{F}^\ell$ for $i \in [n]$. Let $v := (v_1||\ldots||v_n) \in \mathbb{F}^{n\ell}$. If $Av \neq b$, then for any malicious prover, the verifier accepts with probability at most

$$\mathsf{poly}(\lambda)\cdot\left(1/|\mathbb{F}| + ((t+k+\ell)/m)^{|T|} + \epsilon\right),$$

where k is the degree of polynomial p_i that is used to encode v_i to U_i.

Proof. Observe the queries that the cheating prover makes to the random oracle. For each query, consider the following scenario below.

With probability ϵ, the extractor may extract a different matrix from the one actually used by the cheating prover. Therefore, the probability that the verifier accepts is upper bounded by ϵ. Let $U^* \in \mathbb{F}^{n\times m}$ be the matrix formed by rows $U_1^*, \ldots, U_n^*$. In other words, U' is U_0^* juxtaposed on top of U^*.

Conditioned on successful extraction, consider the following case. Let $q(\cdot)$ be the polynomial generated by the honest prover as described in $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$. Note that $Av \neq b$. Consider the probability of a randomly chosen r such that $r^\intercal Av = r^\intercal b$. This means that $r^\intercal(Av-b)=0$, which happens with probability exactly $1/|\mathbb{F}|$.

Let $q^*(\cdot)$ be the polynomial sent by the cheating prover. Consider two cases:

Case 1. $q^*(\cdot)=q(\cdot)$. By the analysis above, the verifier accepts with probability $1/|\mathbb{F}|$.

Case 2. $q^*(\cdot)\neq q(\cdot)$. Since both polynomials have degree $(k+\ell-1)$, the polynomial $q^*(\cdot)-q(\cdot)$ has at most $(k+\ell-1)$ roots. This implies that $q^*(\cdot)$ and $q(\cdot)$ agree on at most $(k+\ell-1)$ points out of $\alpha_1,\ldots,\alpha_m$. Let $A := \{j \in [m] : q^*(\alpha_j)=q(\alpha_j)\}$. Let E be the set of indices of corrupted columns in U' and we know $|E| \leq t$. In order to pass the column check, the random selected subset $T \subseteq (A \cup E)$. This happens with probability

$$\frac{\binom{k+\ell-1+t}{|T|}}{\binom{m}{|T|}} \leq \left(\frac{t+k+\ell}{m}\right)^{|T|}.$$

Finally, we union bound by the (polynomial number of) queries sent by the cheating prover to the random oracle, the probability that the verifier accepts is then upper bounded by $\mathsf{poly}(\lambda)\cdot\left(1/|\mathbb{F}| + ((t+k+\ell)/m)^{|T|} + \epsilon\right)$. □

Taking $k=t+\ell-1$, $\ell=\lambda$, $m=10\lambda$, $t=\lambda$, $|T|=\lambda$, and $\epsilon=\mathsf{negl}(\lambda)$, the soundness of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ is negligible in λ.

Zero-knowledge. For zero-knowledge of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$, the verifier should learn no information about the matrix U (and underlying vector v) except that the linear relationship $Av=b$ holds. To prove this, consider a different vector $v^* \in \mathbb{F}^{n\ell}$ such that $Av^*=b$ and let U^* encode v^*. The verifier's view should be indistinguishable whether U or U^* is used. We state the zero-knowledge lemma below.

Lemma 6.6 (Zero-knowledge of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$)**.** *Assume the verifier can make at most* $\mathsf{poly}(\lambda)$ *queries to the random oracle* $\mathcal{H}$*. The non-interactive argument system* $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ *is zero-knowledge.*

Proof. Consider a simulator S that works as follows:

1. It observes the queries the verifier sends to the random oracle.
2. It samples a random subset $T^* \subseteq [m]$ where $|T^*| = |T|$. It samples matrix U^* as follows: For $j \in T^*$, it samples random columns $U^*[j] \leftarrow \mathbb{F}^n$. For $j \notin T^*$, columns $U^*[j]$ are all-zero columns.
3. It samples random coefficients $r^* \leftarrow \mathbb{F}^n$. It samples a random degree-$(k+\ell-1)$ polynomial $q^*(\cdot)$ conditioned on $\sum_{c=1}^{\ell} q^*(\beta_c) = \sum_{i\in[n],c\in[\ell]} r^*_{ic} \cdot b_{ic}$. Let $\overline{c}^*$ be the coefficients of $q^*(\cdot)$.
4. It samples $u^*_{0,j}$ for $j \in [m]$ as follows: First, it computes $u^*_{0,j} = q^*(\alpha_j) - \sum_{i=1}^{n} r^*_i(\alpha_j) \cdot u^*_{i,j}$ for $j \in T^*$. Then, conditioned on fixing $u^*_{0,j}$ for $j \in T^*$, we sample $u^*_{0,j}$ such that $\sum_{c=1}^{\ell} p^*_0(\beta_c) = 0$ and $u^*_{0,j} = p^*_0(\alpha_j)$.
5. It commits $\mathsf{com}^*_j = \mathsf{Com}(U^*[j])$ and $\mathsf{com}^{*'}_j = \mathsf{Com}(\mathsf{com}^*_j || u^*_{0,j})$ for all $j \in [m]$. Let $\overline{\mathsf{com}}^* := (\mathsf{com}^*_1, \ldots, \mathsf{com}^*_m)$ and $\overline{\mathsf{com}^{*'}} := (\mathsf{com}^{*'}_1, \ldots, \mathsf{com}^{*'}_m)$.
6. It samples a random salt $s_r \leftarrow \{0,1\}^\lambda$ and it programs the random oracle $H_r(\overline{\mathsf{com}}^*, \overline{\mathsf{com}^{*'}}, s_r)$ to return r^*. If the verifier has already made the same query to the random oracle, then abort.
7. It samples a random salt $s_T \leftarrow \{0,1\}^\lambda$ and programs the random oracle $H_T(\overline{\mathsf{com}}^*, \overline{\mathsf{com}^{*'}}, \overline{c}^*, s_T)$ to return T^*. If the same query has already been made by the verifier, abort.
8. It computes the openings $\mathsf{open}^{*'}_j$ and open^*_j for all $j \in T^*$. Specifically, $\mathsf{open}^{*'}_j$ should contain $(\mathsf{com}^*_j || u^*_{0,j})$ and open^*_j should contain $U^*[j]$.
9. It sends $\pi^*_{\mathsf{lin}} := \left(\overline{\mathsf{com}^{*'}}, s_r, s_T, \overline{c}^*, T^*, \{\mathsf{open}^*{}_j : j \in T^*\}, \{\mathsf{open}^{*'}{}_j : j \in T^*\}\right)$.

Consider the following hybrids:

1. $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution where the verifier interacts the prover in $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$.
2. $\underline{\mathsf{Hyb}_1}$: In this hybrid, when the verifier queries the random oracle H_T, we return T^* sampled by S. If the verifier has already made the same query, we abort. Since s_T sampled by the simulator has λ bits, the probability it aborts is at most $2^{-\lambda}$. Note that $|T| \le t$. By the t-privacy of the packed secret sharing scheme, we have $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.
3. $\underline{\mathsf{Hyb}_2}$: In this hybrid, when the verifier queries the random oracle H_r, we return r^* sampled by S. We compute $(r^*)^\intercal A = (r^*_{11}, \ldots, r^*_{1\ell}, \ldots, r^*_{n1}, \ldots, r^*_{n\ell})$. For $i \in [n]$, let $r^*_i(\cdot)$ be the unique polynomial of degree $\ell - 1$ such that $r^*_i(\beta_c) = r^*_{ic}$ for $c \in [\ell]$.

   Since r^* is sampled uniformly at random, it is identically distributed as the output of the random oracle if not previously queried. If the verifier has already made the same query, we abort. Since s_r sampled by S has λ bits, the probability it aborts is at most $2^{-\lambda}$. Thus, we have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

4. $\underline{\mathsf{Hyb}_3}$: For $j \in T^*$, we replace the column $U[j]$ with random columns $U^*[j] \leftarrow \mathbb{F}^n$. We sample a random degree-$(k+\ell-1)$ polynomial $q^*(\cdot)$ conditioned on $\sum_{c=1}^{\ell} q^*(\beta_c) = \sum_{i\in[n],c\in[\ell]} r^*_{ic} \cdot b_{ic}$. For each $j \in T^*$, we compute $u^*_{0,j} = q^*(\alpha_j) - \sum_{i=1}^{n} r^*_i(\alpha_j) \cdot u^*_{i,j}$. Observe that $u^*_{0,j}$ is also uniformly at random. Then, for each $j \in T^*$, we compute commitments $\mathsf{com}^* = \mathsf{Com}(U^*[j])$ and $\mathsf{com}^{*'} = \mathsf{Com}(\mathsf{com}^*||u^*_{0,j})$.

   Note that $|T^*| \le t$. By the t-privacy of the packed secret sharing scheme, the opened values $U[j]$ and $u_{0,j}$ are identically distributed as the random columns $U^*[j]$ and $u^*_{0,j}$ for $j \in T^*$. By the hiding property of the commitment scheme, the commitments between this hybrid and previous hybrid on columns $j \in T^*$ are computationally indistinguishable. Therefore, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_3$.

5. $\underline{\mathsf{Hyb}_4}$: For $j \notin T^*$, we set columns $U^*[j]$ to be all-zero vector. Conditioned on fixing $u^*_{0,j}$ for $j \in T^*$ in the previous hybrid, we sample $u^*_{0,j}$ for $j \notin T^*$ such that $\sum_{c=1}^{\ell} p^*_0(\beta_c) = 0$ and $u^*_{0,j} = p^*_0(\alpha_j)$. Next, for $j \notin T^*$, we commit $\mathsf{com}^* = \mathsf{Com}(U^*[j])$ and $\mathsf{com}^{*'} = \mathsf{Com}(\mathsf{com}^*||u^*_{0,j})$. Note that for $j \notin T^*$, $U^*[j]$ and $u^*_{0,j}$ are not revealed to the verifier. The equation $\sum_{c=1}^{\ell} q^*(\beta_c) = \sum_{c=1}^{\ell} p^*_0(\beta_c) + \sum_{i\in[n],c\in[\ell]} r^*_{ic} \cdot b_{ic} = \sum_{i\in[n],c\in[\ell]} r^*_{ic} \cdot b_{ic}$ still holds since we sample p^*_0 conditioned on $\sum_{c=1}^{\ell} p^*_0(\beta_c) = 0$. By the hiding property of the commitment scheme, we have $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

Note that Hyb_4 corresponds to the simulator. Hence, the verifier's view in the real world is indistinguishable from its view when interacting with S. □

6.3 The Proof System for Boolean Constraints

There is another constraint that needs to be addressed. In the BMR-style garbling used in the outer protocol Φ_{circ} for computing Boolean circuits, both the sender and the receiver generate masking bits for every wire in C. However, the protocol performs arithmetic operations over the finite field $\mathbb{F}$. Consequently, each masking bit must be converted to an element of $\mathbb{F}$. Similarly, the circuit input bits must also be converted into field elements of $\mathbb{F}$. Since they are bits, they should be mapped to 0/1 in $\mathbb{F}$. Following similar ideas as in the proof system for linear constraints, we apply Ligero proofs to construct a proof system $(P, V)^{\mathcal{H}}_{\mathsf{bool}}$ that ensures the Boolean constraints.

A malicious adversary may not perform this field conversion honestly. Therefore, the client needs to prove that locations in vectors $\overline{p}^x, \overline{p}^x_1, \overline{p}^x_2, \overline{p}^x_3, \overline{p}^y, \overline{p}^y_1, \overline{p}^y_2, \overline{p}^y_3$ that correspond to the masking bits or circuit input bits to have value either 0 or 1. To test if a variable x has value either 0 or 1, we can use a quadratic equation $x(1-x) = 0$ over $\mathbb{F}$.

Recall the definition of vector $v := (v_1||\ldots||v_n)$ in the previous section. For $i \in [n]$, recall that $p_i(\cdot)$ is the polynomial used to encode vector $v_i \in \mathbb{F}^\ell$ into a codeword $U_i \in \mathsf{RS}[\mathbb{F}, k, m]$. For $j \in [\ell]$, let $v_{i,j}$ denote the j-th element of v_i, where $v_{i,j} = p_i(\beta_j)$. Let

$$V := \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} v_{1,1} & v_{1,2} & \ldots & v_{1,\ell} \\ v_{2,1} & v_{2,2} & \ldots & v_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n,1} & v_{n,2} & \ldots & v_{n,\ell} \end{pmatrix}$$

be an $n \times \ell$ matrix over $\mathbb{F}$.

The prover wants to prove that for the vectors $v_1, \ldots, v_n$ recovered from the shares represented by matrix U, if for some $i \in [n]$ and $j \in [\ell]$, $v_{i,j}$ represents a masking bit or a circuit input bit, then it must have value either 0 or 1. Recall that U is the matrix

$$U = \begin{pmatrix} \mathsf{Share}(v_1) \\ \mathsf{Share}(v_2) \\ \vdots \\ \mathsf{Share}(v_n) \end{pmatrix} = \begin{pmatrix} u_{1,1} & u_{1,2} & \ldots & u_{1,m} \\ u_{2,1} & u_{2,2} & \ldots & u_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n,1} & u_{n,2} & \ldots & u_{n,m} \end{pmatrix} = \begin{pmatrix} p_1(\alpha_1) & p_1(\alpha_2) & \ldots & p_1(\alpha_m) \\ p_2(\alpha_1) & p_2(\alpha_2) & \ldots & p_2(\alpha_m) \\ \vdots & \vdots & \ddots & \vdots \\ p_n(\alpha_1) & p_n(\alpha_2) & \ldots & p_n(\alpha_m) \end{pmatrix}.$$

Let $\mathbf{1} \in \mathbb{F}^{n \times m}$ denote a matrix consisting of all ones. Define matrix

$$U^c := \mathbf{1} - U = \begin{pmatrix} 1 - u_{1,1} & \ldots & 1 - u_{1,m} \\ \vdots & \ddots & \vdots \\ 1 - u_{n,1} & \ldots & 1 - u_{n,m} \end{pmatrix} = \begin{pmatrix} 1 - p_1(\alpha_1) & \ldots & 1 - p_1(\alpha_m) \\ \vdots & \ddots & \vdots \\ 1 - p_n(\alpha_1) & \ldots & 1 - p_n(\alpha_m) \end{pmatrix},$$

then we can see that U^c encodes matrix

$$V^c := \begin{pmatrix} 1 - v_{1,1} & 1 - v_{1,2} & \ldots & 1 - v_{1,\ell} \\ 1 - v_{2,1} & 1 - v_{2,2} & \ldots & 1 - v_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ 1 - v_{n,1} & 1 - v_{n,2} & \ldots & 1 - v_{n,\ell} \end{pmatrix}.$$

Recall that $\odot$ is the entry-wise product operation. Let

$$U^p := U \odot U^c = \begin{pmatrix} u_{1,1}(1 - u_{1,1}) & u_{1,2}(1 - u_{1,2}) & \ldots & u_{1,m}(1 - u_{1,m}) \\ u_{2,1}(1 - u_{2,1}) & u_{2,2}(1 - u_{2,2}) & \ldots & u_{2,m}(1 - u_{2,m}) \\ \vdots & \vdots & \ddots & \vdots \\ u_{n,1}(1 - u_{n,1}) & u_{n,2}(1 - u_{n,2}) & \ldots & u_{n,m}(1 - u_{n,m}) \end{pmatrix}.$$

Thus, U^p encodes the matrix

$$V^p := V \odot V^c = \begin{pmatrix} v_{1,1}(1 - v_{1,1}) & v_{1,2}(1 - v_{1,2}) & \ldots & v_{1,\ell}(1 - v_{1,\ell}) \\ v_{2,1}(1 - v_{2,1}) & v_{2,2}(1 - v_{2,2}) & \ldots & v_{2,\ell}(1 - v_{2,\ell}) \\ \vdots & \vdots & \ddots & \vdots \\ v_{n,1}(1 - v_{n,1}) & v_{n,2}(1 - v_{n,2}) & \ldots & v_{n,\ell}(1 - v_{n,\ell}) \end{pmatrix}.$$

Let V_i^p denote the i-th row of V^p. Let $V_{i,j}^p$ denote the entry in the i-th row and j-th column of V^p. Observe that if $v_{i,j}$ represents a masking bit or a circuit input bit, then in V^p, it must satisfy $V_{i,j}^p = 0$. Let $v_{\mathsf{bool}} := (V_1^p || \ldots || V_n^p)^\intercal \in \mathbb{F}^{n\ell}$ denote the column vector concatenated by the rows of V^p. Let $A_{\mathsf{bool}} \in \mathbb{F}^{n\ell \times n\ell}$ be a matrix such that $A_{\mathsf{bool}} v_{\mathsf{bool}} = 0$ (where $0 \in \mathbb{F}^{n\ell}$ is the all-zero column vector) describes the linear constraint that if $v_{i,j}$ represents a masking bit or a circuit input bit, then $V_{i,j}^p = 0$. Note that there are actually fewer than $n\ell$ elements of v that need to be converted to 0/1 as not every element of v represents a masking bit or a circuit input bit. Therefore, we do not actually need all $n\ell$ rows in A_{bool}. To fix the issue, we can set the extra rows of A_{bool} as dummy rows of zeros. Following a very similar construction to $(P, V)_{\mathsf{lin}}^{\mathcal{H}}$ in the previous section, we can then test that v_{bool} satisfies the linear constraints given by $A_{\mathsf{bool}} v_{\mathsf{bool}} = 0$. Note that A_{bool} constrains only the positions of masking bits and circuit input bits in the vector. Therefore, the verifier also knows A_{bool} as the prover's vectors are generated by the decomposition agreed by both parties.

We formally describe the non-interactive argument system for Boolean constraints as $(P, V)_{\mathsf{bool}}^{\mathcal{H}}$

in Fig. 11 below. We remark that we need to additionally run $(P,V)^{\mathcal{H}}_{\mathsf{rs}}$ on the matrix U' to ensure it is t-close to $(n+1)$ interleaved RS codewords in L^{n+1}, where U' is defined as prover's matrix U and its additional row U_0 used for masking.

$\underline{(P,V)^{\mathcal{H}}_{\mathsf{bool}}}$:

- **Input**: The purported shares represented as matrix U and their commitments $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$. The witnesses $(p_1, \ldots, p_n)$ which are polynomials of degree k that encode v_i to U_i. Note that $\overline{\mathsf{com}}$ and $A_{\mathsf{bool}} \in \mathbb{F}^{n\ell \times n\ell}$ are the common input to both the prover and the verifier.
- **Prover** $P^{\mathcal{H}}_{\mathsf{bool}}(\overline{\mathsf{com}}, U, (p_1, \ldots, p_n), A_{\mathsf{bool}})$:
  1. It samples a random degree-$(2k+\ell-1)$ polynomial $p_0(\cdot)$ under the constraint that $\sum_{c=1}^{\ell} p_0(\beta_c) = 0$. Let $v_0 := (p_0(\beta_1), \ldots, p_0(\beta_\ell))$ and let $U_0 := (p_0(\alpha_1), \ldots, p_0(\alpha_m)) = (u_{0,1}, \ldots, u_{0,m})$. Then for each $j \in [m]$, it computes $\mathsf{com}'_j := \mathsf{Com}(\mathsf{com}_j || u_{0,j})$. Let $\overline{\mathsf{com}}' := (\mathsf{com}'_1, \ldots, \mathsf{com}'_m)$.
  2. It samples a random salt $s_r \in_R \{0,1\}^\lambda$. Then, it uses the random oracle to generate a column vector $r := H_r(\overline{\mathsf{com}}, \overline{\mathsf{com}}', s_r)$ where $r \in \mathbb{F}^{n\ell}$.
  3. It computes $r^\intercal A_{\mathsf{bool}} = (r_{11}, \ldots, r_{1\ell}, \ldots, r_{n1}, \ldots, r_{n\ell})$. For $i \in [n]$, let $r_i(\cdot)$ be the unique polynomial of degree $\ell - 1$ such that $r_i(\beta_c) = r_{ic}$ for $c \in [\ell]$.
  4. (Folding) It computes a degree-$(2k+\ell-1)$ polynomial $q(\cdot) = p_0(\cdot) + \sum_{i=1}^{n} r_i(\cdot) \cdot p_i(\cdot) \cdot (1 - p_i(\cdot))$. Denote the coefficients of $q(\cdot)$ by vector $\overline{c} = (c_1, \ldots, c_{2k+\ell}) \in \mathbb{F}^{2k+\ell}$.
  5. It samples a random salt $s_T \in_R \{0,1\}^\lambda$. Then it uses the random oracle to choose a subset $T := H_T(\overline{\mathsf{com}}, \overline{\mathsf{com}}', \overline{c}, s_T) \subseteq [m]$ where $|T| = \Theta(m)$.
  6. For each $j \in T$, it computes the openings $\mathsf{open}'_j := \mathsf{Open}(\mathsf{com}'_j)$ and $\mathsf{open}_j := \mathsf{Open}(\mathsf{com}_j)$. More specifically, open'_j should contain $(\mathsf{com}_j || u_{0,j})$ and open_j should contain $U[j]$.
  7. It outputs $\pi_{\mathsf{bool}} := \big(\overline{\mathsf{com}}', s_r, s_T, \overline{c}, T, \{\mathsf{open}'_j : j \in T\}, \{\mathsf{open}_j : j \in T\}\big)$.
- **Verifier** $V^{\mathcal{H}}_{\mathsf{bool}}(\overline{\mathsf{com}}, A_{\mathsf{bool}}, \pi_{\mathsf{bool}})$:
  1. It calls $H_T(\overline{\mathsf{com}}, \overline{\mathsf{com}}', \overline{c}, s_T)$ and checks if it is equal to T sent by the prover. Then, it checks for each $j \in T$ if the openings are successful, i.e. open'_j should successfully open $(\mathsf{com}_j || u_{0,j})$ and open_j should successfully open $U[j]$.
  2. It calls the random oracle $H_r(\overline{\mathsf{com}}, \overline{\mathsf{com}}', s_r)$ to obtain the column vector r. Then it computes $r^\intercal A_{\mathsf{bool}} = (r_{11}, \ldots, r_{1\ell}, \ldots, r_{n1}, \ldots, r_{n\ell})$. For $i \in [n]$, let $r_i(\cdot)$ be the unique polynomial of degree $\ell - 1$ such that $r_i(\beta_c) = r_{ic}$ for $c \in [\ell]$.
  3. It recovers the polynomial $q(\cdot)$ from $\overline{c}$ and does the following:
     (a) For each $j \in T$, it computes $u_{0,j} + \sum_{i=1}^{n} r_i(\alpha_j) \cdot u_{i,j} \cdot (1 - u_{i,j})$ and checks if it is equal to $q(\alpha_j)$ sent by the prover.
     (b) It checks if $\sum_{c=1}^{\ell} q(\beta_c) = 0$.
  4. It accepts if and only if all the checks above pass.

Figure 11: Description of $(P,V)^{\mathcal{H}}_{\mathsf{bool}}$ for Boolean Constraints

Communication Complexity. Consider the message sent by the prover in $(P,V)^{\mathcal{H}}_{\mathsf{bool}}$:

- $\overline{\mathsf{com}}'$: $O(m\lambda)$ bits.
- s_r, s_T: 2λ bits.
- $\overline{c}$: $(2k+\ell)\log|\mathbb{F}| = (2k+\ell)\lambda$ bits.
- T: $|T|\log m$ bits.
- $\{\mathsf{open}'_j : j \in T\}$: $|T| \cdot O(\lambda)$ bits.
- $\{\mathsf{open}_j : j \in T\}$: $|T| \cdot O(n\lambda)$ bits

The total communication is $O(m\lambda) + 2\lambda + (2k+\ell)\lambda + |T| \log m + |T| \cdot O(\lambda) + |T| \cdot O(n\lambda)$, which is dominated by the $|T| \cdot O(n\lambda)$ term. Recall that $n \leq 3N'$ and $N' \cdot \ell = N_{\mathsf{mono}} = O(|C|)$. Thus, the communication complexity is $|T| \cdot O(|C|\lambda/\ell)$. If we take $\ell = \lambda$, $|T| = \Theta(m)$, $m = 8\lambda$, then the communication complexity is $O(|C|\lambda)$.

Completeness. The completeness follows from the description of $(P,V)^{\mathcal{H}}_{\mathsf{bool}}$ and the analysis we gave above. If the field conversion is performed honestly, then the positions of masking bits and circuit input bits in vector v will have value 0 or 1. Hence, the corresponding coordinates of $v_{\mathsf{bool}} = v \odot (1-v)$ are zero, and $A_{\mathsf{bool}} v_{\mathsf{bool}} = 0$.

Soundness. The soundness proof is similar to that of $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$. We state the soundness lemma below.

Lemma 6.7 (Soundness of $(P,V)^{\mathcal{H}}_{\mathsf{bool}}$)**.** *Let $\mathbb{F}$ be a finite field with size 2^λ. Fix some integer $t \in (0, (m-k)/2)$ where $t = \Theta(m)$ and $m = \Theta(\lambda)$. Take $|T| = \Theta(t)$. Let ϵ be the error probability of the extractor Ext of commitment scheme $(\mathsf{Com}, \mathsf{Open})$, where $\epsilon = \mathsf{negl}(\lambda)$. Let $\mathbf{1} \in \mathbb{F}^{n \times m}$ denote the all-one matrix.*

Assume the cheating prover can make at most $\mathsf{poly}(\lambda)$ queries to the random oracle $\mathcal{H}$. Let $U' \in \mathbb{F}^{(n+1)\times m}$ be the matrix extracted by the extractor Ext, and the rows of U' are $U'_0, U'_1, \ldots, U'_n$. Suppose U' is t-close to a codeword $W \in L^{n+1}$, where W contains the codeword $U_0 \in L$ vertically juxtaposed on top of n interleaved codewords $U \in L^n$, and each row $U_i \in \mathbb{F}^m$ encodes $v_i \in \mathbb{F}^\ell$ for $i \in [n]$. Let $v := (v_1 || \ldots || v_n) \in \mathbb{F}^{n\ell}$ and let $v_{\mathsf{bool}} := v \odot (1-v)$ where $1 \in \mathbb{F}^{n\ell}$ denotes all one vector. If $A_{\mathsf{bool}} v_{\mathsf{bool}} \neq 0$, then for any malicious prover, the verifier accepts with probability at most

$$\mathsf{poly}(\lambda) \cdot \left(1/|\mathbb{F}| + ((t+2k+\ell)/m)^{|T|} + \epsilon\right),$$

where k is the degree of polynomial p_i that is used to encode v_i to U_i.

Proof. Observe the queries that the cheating prover makes to the random oracle. For each query, consider the following scenario below.

With probability ϵ, the extractor may extract a different matrix from the one actually used by the cheating prover. Therefore, the probability that the verifier accepts is upper bounded by ϵ. Let $U^* \in \mathbb{F}^{n\times m}$ be the matrix formed by rows $U'_1, \ldots, U'_n$. In other words, U' is U'_0 juxtaposed on top of U^*.

Conditioned on successful extraction, consider the following case. Let $q(\cdot)$ be the polynomial generated by the honest prover as described in $(P,V)^{\mathcal{H}}_{\mathsf{bool}}$. Note that $A_{\mathsf{bool}} v_{\mathsf{bool}} \neq 0$ where $v_{\mathsf{bool}} = v \odot (1-v)$. Consider the event of a randomly chosen vector $r \in \mathbb{F}^{n\ell}$ such that $r^\intercal A_{\mathsf{bool}} v_{\mathsf{bool}} = 0$. This happens with probability exactly $1/|\mathbb{F}|$.

Let $q^*(\cdot)$ be the polynomial sent by the cheating prover. Consider two cases:

Case 1. $q^*(\cdot) = q(\cdot)$. By the analysis above, the verifier accepts with probability $1/|\mathbb{F}|$.

Case 2. $q^*(\cdot) \neq q(\cdot)$. Since both polynomials have degree $(2k+\ell-1)$, the polynomial $q^*(\cdot) - q(\cdot)$ has at most $(2k+\ell-1)$ roots. This implies that $q^*(\cdot)$ and $q(\cdot)$ agree on at most $(2k+\ell-1)$ points out of $\alpha_1, \ldots, \alpha_m$. Let $A := \{j \in [m] : q^*(\alpha_j) = q(\alpha_j)\}$. Let E be the set of indices of corrupted columns in U' and we know $|E| \leq t$. In order to pass the column check, the random selected subset

$T \subseteq (A \cup E)$. This happens with probability

$$\frac{\binom{2k+\ell-1+t}{|T|}}{\binom{m}{|T|}} \leq \left(\frac{t+2k+\ell}{m}\right)^{|T|}.$$

Finally, we union bound by the (polynomial number of) queries sent by the cheating prover to the random oracle, the probability that the verifier accepts is then upper bounded by $\mathsf{poly}(\lambda) \cdot \big(1/|\mathbb{F}| + ((t+2k+\ell)/m)^{|T|} + \epsilon\big)$. □ □

Taking $k = t + \ell - 1$, $\ell = \lambda$, $m = 10\lambda$, $t = \lambda$, $|T| = \lambda$, and $\epsilon = \mathsf{negl}(\lambda)$, the soundness of $(P, V)^{\mathcal{H}}_{\mathsf{bool}}$ is negligible in λ.

Zero-knowledge. We state the zero-knowledge lemma below.

Lemma 6.8 (Zero-knowledge of $(P, V)^{\mathcal{H}}_{\mathsf{bool}}$)**.** *Assume the verifier can make at most* $\mathsf{poly}(\lambda)$ *queries to the random oracle* $\mathcal{H}$*. The non-interactive argument system* $(P, V)^{\mathcal{H}}_{\mathsf{bool}}$ *is zero-knowledge.*

The proof is similar to that of $(P, V)^{\mathcal{H}}_{\mathsf{lin}}$ so we omit the proof to avoid redundancy. The difference is that we sample the random degree-$(2k+\ell-1)$ polynomial $q^*(\cdot)$ conditioned on $\sum_{c=1}^{\ell} q^*(\beta_c) = 0$.

7 The Protocol Compiler

In this section, we use the IPS-style watchlist to compile the outer protocol Φ_{circ} in Section 4 with the inner protocol in Section 5 and the succinct non-interactive arguments in Section 6 in the random oracle model.

Theorem 7.1. *Let* λ *be the security parameter. Let* $\mathbb{F}$ *be a finite field of order* 2^λ*. Let* f *be an arbitrary two-party, single-output functionality modeled by a Boolean circuit* C *composed of XOR and AND gates of fan-in* 2*. Assume the existence of:*

- *A two-round, 2-client, m-server outer protocol* Φ_{circ} *for computing* f *that is secure against an admissible adversary corrupting either client and up to* t *servers, where* t *is a small constant fraction of* m *and* $m = \Theta(\lambda)$*. The outer protocol uses packed Shamir secret sharing of packing size* $\ell = \Theta(\lambda)$*.*
- *A two-round, semi-honest inner protocol* Π_{OLE} *that computes* $k = O(|C|/\ell)$ *instances of OLE over* $\mathbb{F}$ *in parallel, assuming* $O(k)$ *preprocessed random OLE correlations over* $\mathbb{F}$*. Or, equivalently, a two-round, semi-honest inner protocol* Π^*_{OLE} *that computes* k *instances of OLE over* $\mathbb{F}$ *in parallel, assuming* $O(k\lambda)$ *preprocessed random bit-OT correlations.*

Then, there exists a NISC protocol Γ *for computing* f *that makes black-box use of* Π_{OLE} *(or* Π^*_{OLE}*) and is secure against static, malicious adversaries in the random oracle model. The communication complexity of* Γ *is* $O(|C|\lambda)$*.*

7.1 Construction

Building Blocks. The construction makes use of the following building blocks.

1. A conforming outer protocol $\Phi_{\mathsf{circ}} = ((\mathsf{Share}_1, \mathsf{Share}_2), \mathsf{Eval}, \mathsf{Dec})$ for computing f that is secure against an admissible adversary corrupting either client and up to t servers. The $\mathsf{Share}_1, \mathsf{Share}_2$ algorithms make use of packed Shamir secret sharing of packing size ℓ.

2. A semi-honest inner protocol $\Pi_{\mathsf{OLE}} = (\Pi_{\mathsf{OLE},1}, \Pi_{\mathsf{OLE},2}, \mathsf{out}_{\mathsf{OLE}})$ that computes $k = N' = O(|C|/\ell)$ instances of OLE in parallel from OLE correlations. Or equivalently, a two-round, semi-honest inner protocol $\Pi^*_{\mathsf{OLE}} = (\Pi_{\mathsf{OLE},1}, \Pi_{\mathsf{OLE},2}, \mathsf{out}_{\mathsf{OLE}})$ from $O(k\lambda)$ random bit-OT correlations.

3. A straight-line extractable commitment scheme $(\mathsf{Com}, \mathsf{Open})$ constructed unconditionally in the random oracle model [Pas03].

4. The random oracle modeled as hash functions $\mathcal{H} = (H_{\mathsf{com}}, H_T, H_r)$.

5. Non-interactive arguments $(P,V)^{\mathcal{H}}_{\mathsf{rs}}, (P,V)^{\mathcal{H}}_{\mathsf{lin}}, (P,V)^{\mathcal{H}}_{\mathsf{bool}}$ based on Ligero.

Description of the Protocol. We describe a two-party, two-round, malicious-secure protocol $\Gamma(C,x,y)$ for computing f in Fig. 12 below. Note that the openings $\mathsf{Open}(\mathsf{com}^x_j)$ contains $\hat{x}_j$, $\mathsf{Open}(\mathsf{com}^U_j)$ contains $U[j]$, and $\mathsf{Open}(\mathsf{com}^x_{\mathsf{OLE},\mathsf{j}})$ contains $r_{\mathsf{rec},\mathsf{j}}$. Similarly, the openings $\mathsf{Open}(\mathsf{com}^y_j)$ contains $\hat{y}_j$, $\mathsf{Open}(\mathsf{com}^V_j)$ contains $V[j]$, $\mathsf{Open}(\mathsf{com}^{V^0}_j)$ contains $V^0[j]$, and $\mathsf{Open}(\mathsf{com}^y_{\mathsf{OLE},\mathsf{j}})$ contains $r_{\mathsf{sen},\mathsf{j}}$. A concrete parameter choice for the whole protocol is $t = 10\lambda, \ell = \lambda, m = 100\lambda$.

Communication Complexity. Consider the transcript $\mathbb{T}_x, \mathbb{T}_y$ exchanged in the protocol. Take $\ell = \Theta(\lambda)$, $|T_x| = |T_y| = \lambda$, and $m = \Theta(\lambda)$. Consider each component in the transcript: The commitments for the shares $\overline{\mathsf{com}}^x$ (resp. $\overline{\mathsf{com}}^y$) take $m \cdot O(\lambda) = O(\lambda^2)$ bits. The commitments for the shares $\overline{\mathsf{com}}^U$ (resp. $\overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}$) take $m \cdot O(\lambda) = O(\lambda^2)$ bits. The commitments for the correlated randomness $\overline{\mathsf{com}}^x_{\mathsf{OLE}}$ (resp. $\overline{\mathsf{com}}^y_{\mathsf{OLE}}$) take $m \cdot O(\lambda) = O(\lambda^2)$ bits. The proofs $\pi^x_{\mathsf{proofs}} = (\pi^x_{\mathsf{rs}}, \pi^x_{\mathsf{lin}}, \pi^x_{\mathsf{bool}})$ (resp. $\pi^y_{\mathsf{proofs}} = (\pi^y_{\mathsf{rs}}, \pi^y_{\mathsf{lin}}, \pi^y_{\mathsf{bool}})$) have communication complexity $O(|C|\lambda)$. The output of inner protocol π^x_{OLE} (resp. π^y_{OLE}) takes $O(mN'\lambda) = O(m|C|\lambda/\ell) = O(|C|\lambda)$ bits. The random salt tag^x (resp. tag^y) takes λ bits. The random subset T_x (resp. T_y) takes $\lambda \cdot \log m = O(\lambda \log \lambda)$ bits. The openings $\overline{\mathsf{open}}^x$ (and resp. $\overline{\mathsf{open}}^y$) for the shares and the correlated randomness take $\lambda \cdot O(N'\lambda) = O(|C|\lambda)$ bits. Summing them up, the total communication complexity is $O(|C|\lambda) + O(\lambda^2) + O(\lambda \log \lambda)$, which is dominated by $O(|C|\lambda)$ assuming $|C|$ is a (relatively large) polynomial of λ.

Theorem 7.2. *Let λ be the security parameter. Let $f(x,y)$ be an arbitrary two-party, single-output functionality modeled by a Boolean circuit C. Assuming $O(|C|\lambda)$ instances of random bit-OT correlations, the compiled two-party, two-round protocol Γ for computing $f(x,y)$, is secure against malicious adversaries in the random oracle model. The communication complexity of Γ is $O(|C|\lambda)$.*

Security Proof. The security analysis is very similar to that of [IKSS22a]. Let $\mathcal{A}$ be the malicious adversary that is corrupting either the receiver or the sender. Let $\mathsf{Sim}_\Gamma = (\mathsf{Sim}^r_\Gamma, \mathsf{Sim}^s_\Gamma)$ be the simulator in the ideal world. Let $C(x,y)$ be the ideal functionality that outputs Boolean circuit evaluation on input x, y. Let $\mathsf{Sim}_\Phi = (\mathsf{Sim}^r_\Phi, \mathsf{Sim}^s_\Phi)$ be the simulators for the outer protocol Φ_{circ}. Let $\mathsf{Sim}_\Pi = (\mathsf{Sim}^r_\Pi, \mathsf{Sim}^s_\Pi)$ be the simulators for the inner protocol Π_{OLE} (or Π^*_{OLE}). Let dummy denote some dummy value. Let $\mathsf{Sim}_\pi = (\mathsf{Sim}_{\mathsf{rs}}, \mathsf{Sim}_{\mathsf{lin}}, \mathsf{Sim}_{\mathsf{bool}})$ be the zero-knowledge simulator of the Ligero proofs.

Note that both Sim^r_Γ and Sim^s_Γ have full access to both receiver's correlated randomness $r_{\mathsf{rec}} = (r_{\mathsf{rec},1}, \ldots, r_{\mathsf{rec},\mathsf{m}})$ and sender's correlated randomness $r_{\mathsf{sen}} = (r_{\mathsf{sen},1}, \ldots, r_{\mathsf{sen},\mathsf{m}})$ from the environment.

For each unique query made by $\mathcal{A}$ to the random oracle, Sim^r_Γ (resp. Sim^s_Γ) samples a uniform random element in the range of the random oracle, and outputs it as the response. When Sim^r_Γ (resp. Sim^s_Γ) needs to generate a query to the random oracle on behalf of honest sender (resp. receiver) client, it checks if $\mathcal{A}$ has already made the same query. If so, it aborts the execution and instructs the ideal functionality to abort.

If at any point, the checks $\mathsf{verifyProofs}$ or $\mathsf{checkWatchlist}$ outputs 0 on $\mathcal{A}$'s message, then Sim^r_Γ (resp. Sim^s_Γ) aborts and sends $\perp$ to the ideal functionality.

The receiver is corrupted. The simulator $\underline{\mathsf{Sim}^r_\Gamma}$ runs as follows:

1. It chooses a uniform random subset T_y of size λ and a uniform random salt $\mathsf{tag}^y \leftarrow \{0,1\}^\lambda$ and programs the random oracle H_T to output T_y when queried by the message generated by the honest sender client. It starts interacting with the simulator Sim^r_Φ for the outer protocol Φ_{circ} by corrupting the receiver client and the set of servers indexed by T_y in the outer protocol.

2. It obtains the first-round message $\mathbb{T}_x$ from the adversary $\mathcal{A}$. Then, it runs $\mathsf{verifyProofs}$ and $\mathsf{checkWatchlist}$ the same way as the protocol and aborts if any of them returns 0. Next, it runs the straight-line extractor for the commitment scheme $(\mathsf{Com}, \mathsf{Open})$ to extract $(\hat{x}_1, \ldots, \hat{x}_m), U, r_{\mathsf{rec}}$ from adversary's commitment $\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}^x_{\mathsf{OLE}}$.

3. It initializes three empty sets I_1, I_2, I_3.

4. It adds indices of the union of corrupted columns in each Ligero proof to I_1. It corrupts the servers in I_1. If $|I_1| > 3\lambda$, it sends $\perp$ to the ideal functionality.

5. Let $I_2 = T_{\mathsf{rs}} \cup T_{\mathsf{lin}} \cup T_{\mathsf{bool}}$, where $T_{\mathsf{rs}}, T_{\mathsf{lin}}, T_{\mathsf{bool}}$ are subsets of servers where $\mathcal{A}$ cheats on the computation of random linear combination in the Ligero proofs. It corrupts the servers in I_2. If $|I_2| > 3\lambda$, it sends $\perp$ to the ideal functionality.

6. (Check Watchlist) For each $j \in [m]$:

   (a) It checks if the correlated randomness $r_{\mathsf{rec,j}}$ extracted is equal to the correlated randomness that the simulator has access to.

   (b) It simulates the inner protocol $\Pi_{OLE,1}(\hat{x}_j, r_{\mathsf{rec,j}})$ and checks if the output is equal to $\pi^j_{\mathsf{OLE},1}$ obtained from $\mathbb{T}_x$ from $\mathcal{A}$.

   (c) It checks if $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2} \odot \hat{x}_{j,3}$.

   If any of the above checks fail, it corrupts server j and adds j to I_3. If $|I_3| > \lambda$, then it sends $\perp$ to the ideal functionality.

7. It obtains first-round messages $\{\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j\}_{j \in T_y}$ sent by the honest sender client to the corrupted servers in T_y. Let $V[j] = \{\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}\}$ and $V^0[j] = \hat{0}_j$ for $j \in T_y$.

8. For $j \in T_y$, it commits $\mathsf{com}^y_j = \mathsf{Com}(\hat{y}_j)$, $\mathsf{com}^V_j = \mathsf{Com}(V[j])$, $\mathsf{com}^{V^0}_j = \mathsf{Com}(V^0[j])$, and $\mathsf{com}^y_{\mathsf{OLE,j}} = \mathsf{Com}(r_{\mathsf{sen,j}})$. For $j \notin T_y$, it uses dummy values for commitments com^y_j, com^V_j, $\mathsf{com}^{V^0}_j$, $\mathsf{com}^y_{\mathsf{OLE,j}}$. Let $\overline{\mathsf{com}}^y := (\mathsf{com}^y_1, \ldots, \mathsf{com}^y_m)$, $\overline{\mathsf{com}}^V := (\mathsf{com}^V_1, \ldots, \mathsf{com}^V_m)$, $\overline{\mathsf{com}}^{V^0} := (\mathsf{com}^{V^0}_1, \ldots, \mathsf{com}^{V^0}_m)$, and $\overline{\mathsf{com}}^y_{\mathsf{OLE}} := (\mathsf{com}^y_{\mathsf{OLE},1}, \ldots, \mathsf{com}^y_{\mathsf{OLE,m}})$.

9. It prepares the openings $\overline{\mathsf{open}}^y = \{\mathsf{Open}(\mathsf{com}_j^y), \mathsf{Open}(\mathsf{com}_j^V), \mathsf{Open}(\mathsf{com}_j^{V^0}), \mathsf{Open}(\mathsf{com}_{\mathsf{OLE,j}}^y)\}_{j\in T_y}$.

10. It runs simulator for Ligero proofs Sim_π to obtain $\pi_{\mathsf{proofs}}^y = (\pi_{\mathsf{rs}}^y, \pi_{\mathsf{lin}}^y, \pi_{\mathsf{bool}}^y)$.

11. For $j \in T_y$, it runs the inner protocol $\pi_{\mathsf{OLE},2}^j = \Pi_{\mathsf{OLE},2}(\pi_{\mathsf{OLE},1}^j, \hat{y}_j, \hat{0}_j, r_{\mathsf{sen,j}})$ using extracted values from the honest sender to servers in T_y. For $j \notin T_y$, it runs the simulator of the inner protocol Sim_Π^r to obtain $\pi_{\mathsf{OLE},2}^j$. Let $\pi_{\mathsf{OLE}}^y := (\pi_{\mathsf{OLE},2}^1, \ldots, \pi_{\mathsf{OLE},2}^m)$. This will be the servers' response when we invoke the simulator for the outer protocol.

12. It provides $\{\hat{x}_j, U[j]\}_{j\notin I_1\cup I_2\cup I_3}$ to the outer protocol simulator Sim_Φ^r as the messages sent by the adversary to the honest servers. Sim_Φ^r queries the ideal functionality. If Sim_Φ^r instructs the ideal functionality to deliver $\perp$, then Sim_Γ^r sends $\perp$ to the ideal functionality. Else, Sim_Φ^r returns the server evaluations $(\phi_1, \ldots, \phi_m)$, which are also the second-round messages of the inner protocol $\pi_{\mathsf{OLE}}^y = (\pi_{\mathsf{OLE},2}^1, \ldots, \pi_{\mathsf{OLE},2}^m) = (\phi_1, \ldots, \phi_m)$.

13. It programs the random oracle to return the sampled T_y when the adversary queries H_T on input $(\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}_{\mathsf{OLE}}^y, \pi_{\mathsf{proofs}}^y, \pi_{\mathsf{OLE}}^y, \mathsf{tag}^y)$.

14. It sends $\mathbb{T}_y := (\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}_{\mathsf{OLE}}^y, \pi_{\mathsf{proofs}}^y, \pi_{\mathsf{OLE}}^y, \mathsf{tag}^y, T_y, \overline{\mathsf{open}}^y)$ to $\mathcal{A}$ and outputs whatever it outputs.

Proof. Consider the following sequence of hybrids:

1. $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution.

2. $\underline{\mathsf{Hyb}_1}$: This hybrid corresponds to the distribution where the random oracle queries by $\mathcal{A}$ are answered with a uniform random element from the range of the oracle. Further, if the adversary makes any queries to the random oracle before the exact same query was made by the honest party, we abort. Specifically, it picks a uniform random subset $T_y \subseteq [m]$ with size λ, and a salt $\mathsf{tag}^y \leftarrow \{0,1\}^\lambda$, and programs the random oracle to return T_y. It corrupts the servers in T_y and obtains the shares sent by the honest sender to servers in T_y. Since each query made to the oracle contains a uniform random salt of length λ, the probability that $\mathcal{A}$ makes a query that exactly matches the string queried by an honest party is at most $q \cdot 2^{-\lambda}$, where $q = \mathsf{poly}(\lambda)$ is the total number of queries made by $\mathcal{A}$ to the random oracle. Therefore, we have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

3. $\underline{\mathsf{Hyb}_2}$: After obtaining the first-round message $\mathbb{T}_x$ from $\mathcal{A}$, we run $\mathsf{verifyProofs}$ and $\mathsf{checkWatchlist}$ the same way as the real execution and aborts if any of them returns 0. Next, we run the straight-line extractor for the commitment scheme to extract $(\hat{x}_1, \ldots, \hat{x}_m), U, r_{\mathsf{rec}}$ from adversary's commitment $\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}_{\mathsf{OLE}}^x$. Since we do not modify any values in this hybrid, we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

4. $\underline{\mathsf{Hyb}_3}$: In this hybrid, we make the following changes:

   (a) It initializes two empty sets I_1, I_2.

   (b) It adds the union of indices of corrupted columns in Ligero proofs to I_1 as described in the simulator, and corrupt the servers in I_1.

(c) It adds the union of indices of servers who cheats in the random linear combination of Ligero proofs to I_2, as described in the simulator, and corrupt the servers in I_2.

If $|I_1| > 3\lambda$ or $|I_2| > 3\lambda$, we abort. It follows from the soundness of Ligero arguments $(P,V)^{\mathsf{H}}_{\mathsf{rs}}$, $(P,V)^{\mathsf{H}}_{\mathsf{lin}}$, $(P,V)^{\mathsf{H}}_{\mathsf{bool}}$ that $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

5. $\underline{\mathsf{Hyb}_4}$: In this hybrid, initialize an empty set I_3. We add the indices of servers to I_3 if any of the conditions below is true:

   (a) The correlated randomness $r_{\mathsf{rec,j}}$ extracted from $\mathcal{A}$'s message is different from the correlated randomness that the simulator Sim^r_Γ has access to.

   (b) The received inner protocol message $\pi^j_{\mathsf{OLE},1}$ is different from the output of the simulator running $\Pi_{OLE,1}(\hat{x}_j, r_{\mathsf{rec,j}})$ by itself.

   (c) The extracted shares $\hat{x}_j \neq \hat{x}_{j,1} \odot \hat{x}_{j,2} \odot \hat{x}_{j,3}$.

   Corrupt the servers in I_3. If $|I_3| > \lambda$, we abort and instruct the ideal functionality to output $\perp$. Note that if $|I_3| > \lambda$, then the randomly selected watchlist T_x fails to catch any server in I_3 with probability at most $\binom{m-\lambda}{\lambda}/\binom{m}{\lambda} = 2^{-O(\lambda)}$ given $m = \Theta(\lambda)$. Thus, we have $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$.

6. $\underline{\mathsf{Hyb}_5}$: In this hybrid, we use the zero-knowledge simulator Sim_π to generate the proofs, and replace $\pi^y_{\mathsf{proofs}} = (\pi^y_{\mathsf{rs}}, \pi^y_{\mathsf{lin}}, \pi^y_{\mathsf{bool}})$ with the simulated proofs. By the zero-knowledge of the proof systems, we have $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

7. $\underline{\mathsf{Hyb}_6}$: In this hybrid, for $j \notin T_y$, we replace the commitments in $\overline{\mathsf{com}}^y$, $\overline{\mathsf{com}}^V$, $\overline{\mathsf{com}}^{V^0}$, $\overline{\mathsf{com}}^y_{\mathsf{OLE}}$ with commitment of dummy values. By the hiding property of the commitment scheme, we have $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

8. $\underline{\mathsf{Hyb}_7}$: In this hybrid, we use the simulator for the inner protocol Sim^r_Π to simulate the executions for server $j \notin T_y$. By the information-theoretic security of the inner protocol, we have $\mathsf{Hyb}_7 \equiv \mathsf{Hyb}_6$.

9. $\underline{\mathsf{Hyb}_8}$: In this hybrid, we provide $\{\hat{x}_j, U[j]\}_{j \notin I_1 \cup I_2 \cup I_3}$ to the outer protocol simulator Sim^r_Φ as the messages sent by the adversary to the honest servers. Note that $|T_y|+|I_1|+|I_2|+|I_3| \leq 8\lambda$. Set the number of corrupted servers $t \geq 8\lambda$, $\ell = \lambda$, and $m = \Theta(\lambda)$ large enough such that t is within the error-correcting distance. Sim^r_Φ queries the ideal functionality. If Sim^r_Φ instructs the ideal functionality to deliver $\perp$, then Sim^r_Γ forwards it to the ideal functionality. Else, Sim^r_Φ returns the server evaluations $\pi^y_{\mathsf{OLE}} = (\pi^1_{\mathsf{OLE},2}, \ldots, \pi^m_{\mathsf{OLE},2}) = (\phi_1, \ldots, \phi_m)$. By the security of the outer protocol, we have $\mathsf{Hyb}_8 \approx_c \mathsf{Hyb}_7$.

Note that Hyb_8 is the ideal execution by the simulator. Hence, the ideal and real executions are computationally indistinguishable. □

The sender is corrupted. The simulator $\underline{\mathsf{Sim}^s_\Gamma}$ runs as follows:

1. It chooses a uniform random subset T_x of size λ and a uniform random salt $\mathsf{tag}^x \in \{0,1\}^\lambda$ and programs the random oracle H_T to output T_x when queried by the message generated by the honest receiver client. It starts interacting with the simulator Sim^s_Φ for the outer protocol Φ_{circ} by corrupting the sender client and the set of servers indexed by T_x in the outer protocol.

2. It obtains first-round messages $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}\}_{j \in T_x}$ sent by the honest receiver client to the corrupted servers in T_x.

3. For $j \in T_x$, it computes $\mathsf{com}_j^x = \mathsf{Com}(\hat{x}_j)$, $\mathsf{com}_j^U = \mathsf{Com}(U[j])$, $\mathsf{com}_{\mathsf{OLE,j}}^x = \mathsf{Com}(r_{\mathsf{rec,j}})$. For $j \notin T_x$, it uses dummy values for commitments com_j^x, com_j^U, $\mathsf{com}_{\mathsf{OLE,j}}^x$. Let $\overline{\mathsf{com}}^x := (\mathsf{com}_1^x, \ldots, \mathsf{com}_m^x)$, $\overline{\mathsf{com}}^U := (\mathsf{com}_1^U, \ldots, \mathsf{com}_m^U)$, and $\overline{\mathsf{com}}_{\mathsf{OLE}}^x := (\mathsf{com}_{\mathsf{OLE},1}^x, \ldots, \mathsf{com}_{\mathsf{OLE,m}}^x)$.

4. It prepares $\overline{\mathsf{open}}^x := \{\mathsf{Open}(\mathsf{com}_j^x), \mathsf{Open}(\mathsf{com}_j^U), \mathsf{Open}(\mathsf{com}_{\mathsf{OLE,j}}^x)\}_{j \in T_x}$.

5. (Simulate Proofs) It uses the zero-knowledge simulator Sim_π to simulate the proofs $\pi_{\mathsf{proofs}}^x = (\pi_{\mathsf{rs}}^x, \pi_{\mathsf{lin}}^x, \pi_{\mathsf{bool}}^x)$.

6. (Simulate Inner Protocol) For $j \in T_x$, it executes the inner protocol $\pi_{\mathsf{OLE},1}^j = \Pi_{\mathsf{OLE},1}(\hat{x}_j, r_{\mathsf{rec,j}})$. For $j \notin T_x$, it uses the simulator for the inner protocol Sim_Π^s to generate $\pi_{\mathsf{OLE},1}^j$. Let $\pi_{\mathsf{OLE}}^x := (\pi_{\mathsf{OLE},1}^1, \ldots, \pi_{\mathsf{OLE},1}^m)$.

7. It programs the random oracle to return the sampled T_x when the adversary queries H_T with input $(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}_{\mathsf{OLE}}^x, \pi_{\mathsf{proofs}}^x, \pi_{\mathsf{OLE}}^x, \mathsf{tag}^x)$.

8. It sends $\mathbb{T}_x := \left(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}_{\mathsf{OLE}}^x, \pi_{\mathsf{proofs}}^x, \pi_{\mathsf{OLE}}^x, \mathsf{tag}^x, T_x, \overline{\mathsf{open}}^x\right)$ to $\mathcal{A}$ on behalf of the honest receiver.

9. It obtains the second-round message $\mathbb{T}_y$ from the adversary $\mathcal{A}$. Then, it runs $\mathsf{verifyProofs}$ and $\mathsf{checkWatchlist}$ the same way as the protocol and aborts if any of them returns 0. Next, it runs the straight-line extractor for the commitment scheme $(\mathsf{Com}, \mathsf{Open})$ to extract $(\hat{y}_1, \ldots, \hat{y}_m), V, V^0, r_{\mathsf{sen}}$ from adversary's commitment $\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}_{\mathsf{OLE}}^y$.

10. It initializes three empty sets I_1, I_2, I_3.

11. It adds indices of the union of corrupted columns in each Ligero proof to I_1. It corrupts the servers in I_1. If $|I_1| > 4\lambda$ (note that there is a separate Ligero proof for additive secret sharing of zeros), it sends $\perp$ to the ideal functionality.

12. Let $I_2 = T_{\mathsf{rs}} \cup T_{\mathsf{lin}} \cup T_{\mathsf{bool}}$, where $T_{\mathsf{rs}}, T_{\mathsf{lin}}, T_{\mathsf{bool}}$ are subsets of servers where $\mathcal{A}$ cheats on the computation of random linear combination in the Ligero proofs. It corrupts the servers in I_2. If $|I_2| > 4\lambda$, it sends $\perp$ to the ideal functionality.

13. (Check Watchlist) For each $j \in [m]$:

    (a) It checks if the correlated randomness $r_{\mathsf{sen,j}}$ extracted is equal to the correlated randomness that the simulator has access to.

    (b) It simulates the inner protocol $\Pi_{\mathsf{OLE},2}(\pi_{\mathsf{OLE},1}^j, \hat{y}_j, \hat{0}_j, r_{\mathsf{sen,j}})$ and checks if the output is equal to $\pi_{\mathsf{OLE},2}^j$ obtained from $\mathbb{T}_y$ from $\mathcal{A}$.

    (c) It checks whether $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2} \odot \hat{y}_{j,3}$.

    If any of the above checks fail, it corrupts server j and adds j to I_3. If $|I_3| > \lambda$, then it sends $\perp$ to the ideal functionality.

14. It provides $\{\hat{y}_j, V[j], V^0[j]\}_{j \notin I_1 \cup I_2 \cup I_3}$ to the outer protocol simulator Sim^s_Φ as the messages sent by $\mathcal{A}$ to the honest servers. Sim^s_Φ queries the ideal functionality:
    - If Sim^s_Φ instructs the ideal functionality to deliver the output to honest receiver, then Sim^s_Γ forwards this message.
    - If Sim^s_Φ instructs the ideal functionality to deliver $\perp$, Sim^s_Γ sends $\perp$ to the ideal functionality.
15. Sim^s_Γ outputs whatever $\mathcal{A}$ outputs.

Proof. Consider the following sequence of hybrids:

1. $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution.
2. $\underline{\mathsf{Hyb}_1}$: This hybrid corresponds to the distribution where the random oracle queries by $\mathcal{A}$ are answered with a uniform random element from the range of the oracle. Further, if the adversary makes any queries to the random oracle before the exact same query was made by the honest party, we abort. Specifically, it picks a uniform random subset $T_x \subseteq [m]$ with size λ, and a salt $\mathsf{tag}^x \leftarrow \{0,1\}^\lambda$, and programs the random oracle to return T_x.

   Since each query made to the oracle contains a uniform random salt of length λ, the probability that $\mathcal{A}$ makes a query that exactly matches the string queried by an honest party is at most $q \cdot 2^{-\lambda}$, where $q = \mathsf{poly}(\lambda)$ is the total number of queries made by $\mathcal{A}$ to the random oracle. Therefore, we have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.
3. $\underline{\mathsf{Hyb}_2}$: In this hybrid, we use the zero-knowledge simulator Sim_π to generate the proofs, and replace $\pi^x_{\mathsf{proofs}} = (\pi^x_{\mathsf{rs}}, \pi^x_{\mathsf{lin}}, \pi^x_{\mathsf{bool}})$ with the simulated proofs. By the zero-knowledge property of the proof systems, we have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.
4. $\underline{\mathsf{Hyb}_3}$: In this hybrid, we corrupt the servers in T_x and thus obtain the first-round messages $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}\}_{j \in T_x}$ sent by the honest receiver client to the corrupted servers in T_x. For $j \notin T_x$, we replace the commitments in $\overline{\mathsf{com}}^x$, $\overline{\mathsf{com}}^U$, $\overline{\mathsf{com}}^x_{\mathsf{OLE}}$ with commitment of dummy values. By hiding property of the commitment scheme, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_3$.
5. $\underline{\mathsf{Hyb}_4}$: In this hybrid, we use the simulator for the inner protocol Sim^s_Π to simulate the executions for server $j \notin T_x$. For $j \in T_x$, we use the values in the real execution to compute $\pi^j_{\mathsf{OLE},1} = \Pi_{\mathsf{OLE},1}(\hat{x}_j, r_{\mathsf{rec,j}})$. By the information-theoretic security of the inner protocol, we have $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_3$.
6. $\underline{\mathsf{Hyb}_5}$: In this hybrid, we program the random oracle to return T_x when the adversary queries H_T on input $(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}^x_{\mathsf{OLE}}, \pi^x_{\mathsf{proofs}}, \pi^x_{\mathsf{OLE}}, \mathsf{tag}^x)$. If $\mathcal{A}$ has made the same query, we abort. Since the salt $\mathsf{tag}^x \in \{0,1\}^\lambda$ is sampled uniformly at random, we have $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$.
7. $\underline{\mathsf{Hyb}_6}$: After obtaining the message $\mathbb{T}_y$ from $\mathcal{A}$, we run $\mathsf{verifyProofs}$ and $\mathsf{checkWatchlist}$ the same way as the real execution and abort if any of them returns 0. Next, we run the straight-line extractor for the commitment scheme to extract $(\hat{y}_1, \ldots, \hat{y}_m), V, V^0, r_{\mathsf{sen}}$ from adversary's commitments $\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}^y_{\mathsf{OLE}}$ respectively. Since we do not modify any values in this hybrid, we have $\mathsf{Hyb}_6 \equiv \mathsf{Hyb}_5$.
8. $\underline{\mathsf{Hyb}_7}$: In this hybrid, we make the following changes:

(a) It initializes two empty sets I_1, I_2.

(b) It adds the union of indices of corrupted columns in Ligero proofs to I_1 as described in the simulator, and corrupt the servers in I_1.

(c) It adds the union of indices of servers on which the adversary cheats in the random linear combination of Ligero proofs to I_2, as described in the simulator, and corrupt the servers in I_2.

If $|I_1| > 4\lambda$ or $|I_2| > 4\lambda$, we abort. It follows from the soundness of Ligero arguments $(P,V)^{\mathsf{H}}_{\mathsf{rs}}$, $(P,V)^{\mathsf{H}}_{\mathsf{lin}}$, $(P,V)^{\mathsf{H}}_{\mathsf{bool}}$ that $\mathsf{Hyb}_7 \approx_c \mathsf{Hyb}_6$.

9. Hyb_8: In this hybrid, initialize an empty set I_3. We add the indices of servers to I_3 if any of the conditions below is true:

   (a) The correlated randomness $r_{\mathsf{sen,j}}$ extracted from $\mathcal{A}$'s message is different from the correlated randomness that the simulator Sim^s_Γ has access to.

   (b) The received inner protocol message $\pi^j_{\mathsf{OLE},2}$ is different from the output of the simulator running $\Pi_{\mathsf{OLE},2}(\pi^j_{\mathsf{OLE},1}, \hat{y}_j, \hat{0}_j, r_{\mathsf{sen,j}})$ by itself.

   (c) The extracted shares $\hat{y}_j \neq \hat{y}_{j,1} \odot \hat{y}_{j,2} \odot \hat{y}_{j,3}$.

   Corrupt the servers in I_3. If $|I_3| > \lambda$, we abort and instruct the ideal functionality to output $\perp$. Note that if $|I_3| > \lambda$, then the randomly selected watchlist T_y fails to catch any server in I_3 with probability at most $\binom{m-\lambda}{\lambda}/\binom{m}{\lambda} = 2^{-O(\lambda)}$ given $m = \Theta(\lambda)$. Thus, we have $\mathsf{Hyb}_8 \approx_c \mathsf{Hyb}_7$.

10. Hyb_9: In this hybrid, we provide $\{\hat{y}_j, V[j], V^0[j]\}_{j \notin I_1 \cup I_2 \cup I_3}$ to the outer protocol simulator Sim^s_Φ as the messages $\mathcal{A}$ sends to the honest servers. Sim^s_Φ queries the ideal functionality:

    - If Sim^s_Φ instructs the ideal functionality to deliver the output to honest receiver, then Sim^s_Γ forwards this message.
    - If Sim^s_Φ instructs the ideal functionality to deliver $\perp$, Sim^s_Γ sends $\perp$ to the ideal functionality.

    By the security of the outer protocol Φ_{circ}, we have $\mathsf{Hyb}_9 \approx_c \mathsf{Hyb}_8$. Note that $|T_x| + |I_1| + |I_2| + |I_3| \leq 10\lambda$. Set the number of corrupted servers $t \geq 10\lambda$, $\ell = \lambda$, and $m = \Theta(\lambda)$ large enough such that t is within the error-correcting distance.

Note that Hyb_9 corresponds to the ideal execution. Hence, the ideal and real executions are computationally indistinguishable. □

8 Concrete Efficiency

In this section, we analyze the concrete communication cost of our NISC protocol and describe some optimizations that can improve the communication cost.

Malicious Two-Party NISC $\Gamma(C, x, y)$:

- **Setup:**
  - The receiver (resp. sender) obtains vector $\overline{x}$ (resp. $\overline{y}$), and as common input linear constraints $\mathcal{R} = ((A^x, b^x), (A^y, b^y))$ and Boolean constraints $\mathcal{B} = (A^x_{\mathsf{bool}}, A^y_{\mathsf{bool}})$ from the setup of outer protocol $\Phi_{\mathsf{circ}}(C, x, y)$.
  - The receiver (resp. sender) obtains correlations $r_{\mathsf{rec}} := (r_{\mathsf{rec},1}, \ldots, r_{\mathsf{rec,m}})$ (resp. $r_{\mathsf{sen}} := (r_{\mathsf{sen},1}, \ldots, r_{\mathsf{sen,m}})$) from the setup of inner protocol Π_{OLE}.
- **The receiver does:**
  1. It obtains $\{\hat{x}_j, \hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}\}_{j\in[m]} = \mathsf{Share}_1(1^\lambda, \overline{x})$. Let $U \in \mathbb{F}^{n\times m}$ be a matrix formed by $\{\hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}\}_{j\in[m]}$.
  2. It commits $\overline{\mathsf{com}}^U := (\mathsf{com}^U_1, \ldots, \mathsf{com}^U_m)$, where $\mathsf{com}^U_j = \mathsf{Com}(U[j])$ for $j \in [m]$. It commits $\overline{\mathsf{com}}^x := (\mathsf{com}^x_1, \ldots, \mathsf{com}^x_m)$, where $\mathsf{com}^x_j = \mathsf{Com}(\hat{x}_j)$ for $j \in [m]$.
  3. It runs $\mathsf{makeProofs}$ (described in Fig. 13) to generate the Ligero proofs for U, denoted by π^x_{proofs}.
  4. It commits $\overline{\mathsf{com}}^x_{\mathsf{OLE}} := (\mathsf{com}^x_{\mathsf{OLE},1}, \ldots, \mathsf{com}^x_{\mathsf{OLE,m}})$, where $\mathsf{com}^x_{\mathsf{OLE,j}} = \mathsf{Com}(r_{\mathsf{rec,j}})$ for $j \in [m]$. Then it obtains first-round messages $\pi^x_{\mathsf{OLE}} := (\pi^1_{\mathsf{OLE},1}, \ldots, \pi^m_{\mathsf{OLE},1})$, where $\pi^j_{\mathsf{OLE},1} = \Pi_{\mathsf{OLE},1}(\hat{x}_j, r_{\mathsf{rec,j}})$ for $j \in [m]$.
  5. It obtains subset $T_x = H_T(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}^x_{\mathsf{OLE}}, \pi^x_{\mathsf{proofs}}, \pi^x_{\mathsf{OLE}}, \mathsf{tag}^x) \subseteq [m]$ where $\mathsf{tag}^x \leftarrow \{0,1\}^\lambda$. It then computes the openings $\overline{\mathsf{open}}^x := \{\mathsf{Open}(\mathsf{com}^x_j), \mathsf{Open}(\mathsf{com}^U_j), \mathsf{Open}(\mathsf{com}^x_{\mathsf{OLE,j}})\}_{j\in T_x}$.
  6. It outputs $\mathbb{T}_x := \left(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}^x_{\mathsf{OLE}}, \pi^x_{\mathsf{proofs}}, \pi^x_{\mathsf{OLE}}, \mathsf{tag}^x, T_x, \overline{\mathsf{open}}^x\right)$.
- **The sender does:**
  1. It runs $\mathsf{verifyProofs}$ (described in Fig. 14) for U and $\mathsf{checkWatchlist}(0, \mathbb{T}_x, r_{\mathsf{sen}})$. If any of them outputs 0, it aborts.
  2. It obtains $\{\hat{y}_j, \hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}, \hat{0}_j\}_{j\in[m]} = \mathsf{Share}_2(1^\lambda, \overline{y})$. Let $V \in \mathbb{F}^{n\times m}$ be a matrix formed by $\{\hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}\}_{j\in[m]}$. Let $V^0 \in \mathbb{F}^{n_0\times m}$ be a matrix formed by $\{\hat{0}_j\}_{j\in[m]}$.
  3. It commits $\overline{\mathsf{com}}^V := (\mathsf{com}^V_1, \ldots, \mathsf{com}^V_m)$, $\overline{\mathsf{com}}^{V^0} := (\mathsf{com}^{V^0}_1, \ldots, \mathsf{com}^{V^0}_m)$, and $\overline{\mathsf{com}}^y := (\mathsf{com}^y_1, \ldots, \mathsf{com}^y_m)$ similarly as above.
  4. It runs $\mathsf{makeProofs}$ to generate the Ligero proofs for V and V^0, denoted by π^y_{proofs}.
  5. It commits $\overline{\mathsf{com}}^y_{\mathsf{OLE}} := (\mathsf{com}^y_{\mathsf{OLE},1}, \ldots, \mathsf{com}^y_{\mathsf{OLE,m}})$, where $\mathsf{com}^y_{\mathsf{OLE,j}} = \mathsf{Com}(r_{\mathsf{sen,j}})$ for $j \in [m]$. Then it gets second-round messages $\pi^y_{\mathsf{OLE}} := (\pi^1_{\mathsf{OLE},2}, \ldots, \pi^m_{\mathsf{OLE},2})$ where $\pi^j_{\mathsf{OLE},2} = \Pi_{\mathsf{OLE},2}(\pi^j_{\mathsf{OLE},1}, \hat{y}_j, \hat{0}_j, r_{\mathsf{sen,j}})$ for $j \in [m]$.
  6. It gets $T_y = H_T(\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}^y_{\mathsf{OLE}}, \pi^y_{\mathsf{proofs}}, \pi^y_{\mathsf{OLE}}, \mathsf{tag}^y) \subseteq [m]$ where $\mathsf{tag}^y \leftarrow \{0,1\}^\lambda$. It then computes the openings $\overline{\mathsf{open}}^y := \{\mathsf{Open}(\mathsf{com}^y_j), \mathsf{Open}(\mathsf{com}^V_j), \mathsf{Open}(\mathsf{com}^{V^0}_j), \mathsf{Open}(\mathsf{com}^y_{\mathsf{OLE,j}})\}_{j\in T_y}$.
  7. It outputs $\mathbb{T}_y := \left(\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}^y_{\mathsf{OLE}}, \pi^y_{\mathsf{proofs}}, \pi^y_{\mathsf{OLE}}, \mathsf{tag}^y, T_y, \overline{\mathsf{open}}^y\right)$.
- **Output:** The receiver does:
  1. It runs $\mathsf{verifyProofs}$ for V, V^0 and $\mathsf{checkWatchlist}(1, \mathbb{T}_y, r_{\mathsf{rec}}, \pi^x_{\mathsf{OLE}})$ (described in Figure 15). If any of them outputs 0, then it aborts and outputs $\perp$.
  2. It obtains $\phi_j = \mathsf{out}_{\mathsf{OLE}}(\pi^j_{\mathsf{OLE},2}, r_{\mathsf{rec,j}})$ for $j \in [m]$.
  3. It runs $\mathsf{Dec}(1^\lambda, (\phi_1, \ldots, \phi_m))$ and outputs whatever it outputs.

Figure 12: Description of Two-Round Malicious 2PC

$\mathsf{makeProofs}(n, U, \overline{\mathsf{com}}, (p_1, \ldots, p_n), A, b, A_{\mathsf{bool}})$:

- **Inputs:**
  1. A parameter n. When we need to prove for $U = \{\hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}\}_{j\in[m]}$ or $V = \{\hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}\}_{j\in[m]}$, then $n = 3N'$. When we need to prove for $V^0 = \{\hat{0}_j\}_{j\in[m]}$, then $n = N'$. Note that $N' = O(|C|/\lambda)$.
  2. A matrix $U \in \mathbb{F}^{n\times m}$ and the commitment of its columns $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$ where com_j is the commitment of $U[j]$ for $j \in [m]$.
  3. An array of polynomials $(p_1, \ldots, p_n)$ used in the packed secret sharing scheme in $\mathsf{Share}_1, \mathsf{Share}_2$ such that row $U_i = (p_i(\alpha_1), \ldots, p_i(\alpha_m))$ for all $i \in [n]$. These polynomials uniquely define the secret vectors $v_i = (p_i(\beta_1), \ldots, p_i(\beta_\ell))$ for all $i \in [n]$. Let $v := v_1||\ldots||v_n$.
  4. A matrix $A \in \mathbb{F}^{n\ell\times n\ell}$ and a vector $b \in \mathbb{F}^{n\ell}$ that describe the linear constraints by $Av = b$.
  5. A matrix $A_{\mathsf{bool}} \in \mathbb{F}^{n\ell\times n\ell}$ that describes the Boolean constraints by $A_{\mathsf{bool}}\,(v \odot (1 - v)) = 0$, where $1 \in \mathbb{F}^{n\ell}$ is the all-one column vector and $0 \in \mathbb{F}^{n\ell}$ is the all-zero column vector. When we need to prove for $\{\hat{0}_j\}_{j\in[m]}$, then A_{bool} is not needed.
- **Execution:**
  1. It runs $\pi_{\mathsf{rs}} := P^{\mathcal{H}}_{\mathsf{rs}}(\overline{\mathsf{com}}, U, (p_1, \ldots, p_n))$ with proximity parameter (number of corrupted columns) $t_{\mathsf{rs}} = \lambda$, and random subset $|T_{\mathsf{rs}}| = \lambda$.
  2. It runs $\pi_{\mathsf{lin}} := P^{\mathcal{H}}_{\mathsf{lin}}(\overline{\mathsf{com}}, U, (p_1, \ldots, p_n), A, b)$ with random subset $|T_{\mathsf{lin}}| = \lambda$.
  3. It runs $\pi_{\mathsf{bool}} := P^{\mathcal{H}}_{\mathsf{bool}}(\overline{\mathsf{com}}, U, (p_1, \ldots, p_n), A_{\mathsf{bool}})$ with random subset $|T_{\mathsf{bool}}| = \lambda$. When we need to prove for $\{\hat{0}_j\}_{j\in[m]}$, this is not needed.
  4. It outputs $\pi_{\mathsf{proofs}} := (\pi_{\mathsf{rs}}, \pi_{\mathsf{lin}}, \pi_{\mathsf{bool}})$.

Figure 13: Description of makeProofs

$\mathsf{verifyProofs}(n, \overline{\mathsf{com}}, A, b, A_{\mathsf{bool}}, \pi_{\mathsf{proofs}} = (\pi_{\mathsf{rs}}, \pi_{\mathsf{lin}}, \pi_{\mathsf{bool}}))$:

- **Context:** The shares sent by the prover are represented by the matrix $U \in \mathbb{F}^{n\times m}$. Let $v := v_1||\ldots||v_n$, where U_i encodes v_i for $i \in [n]$.
- **Inputs:**
  1. A parameter n. For U, V, set $n = 3N'$. For V^0, set $n = N'$.
  2. The commitments of matrix U's columns $\overline{\mathsf{com}} = (\mathsf{com}_1, \ldots, \mathsf{com}_m)$.
  3. A matrix $A \in \mathbb{F}^{n\ell\times n\ell}$ and a vector $b \in \mathbb{F}^{n\ell}$ that describe the linear constraints by $Av = b$.
  4. A matrix $A_{\mathsf{bool}} \in \mathbb{F}^{n\ell\times n\ell}$ that describes Boolean constraints by $A_{\mathsf{bool}}\,(v \odot (1 - v)) = 0$. For V^0, this is not needed.
  5. The Ligero proofs $\pi_{\mathsf{proofs}} = (\pi_{\mathsf{rs}}, \pi_{\mathsf{lin}}, \pi_{\mathsf{bool}})$.
- **Execution:**
  1. It runs verifier's algorithm $V^{\mathcal{H}}_{\mathsf{rs}}(\overline{\mathsf{com}}, \pi_{\mathsf{rs}})$.
  2. It runs verifier's algorithm $V^{\mathcal{H}}_{\mathsf{lin}}(\overline{\mathsf{com}}, A, b, \pi_{\mathsf{lin}})$.
  3. It runs verifier's algorithm $V^{\mathcal{H}}_{\mathsf{bool}}(\overline{\mathsf{com}}, A_{\mathsf{bool}}, \pi_{\mathsf{bool}})$ (not needed for V^0).
  4. If any of the verifiers above rejects, it outputs 0. Else, it outputs 1.

Figure 14: Description of verifyProofs

$\mathsf{checkWatchlist}(k, \mathbb{T}, r_{\mathsf{cor}}, \pi_{\mathsf{OLE}})$:

- **Inputs:**
  1. An index $k \in \{0,1\}$. If $k = 0$, then $\mathbb{T} = \mathbb{T}_x$ is the receiver's message. If $k = 1$, then $\mathbb{T} = \mathbb{T}_y$ is the sender's message.
  2. Transcript
$$\mathbb{T} = \begin{cases} \left(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}^x_{\mathsf{OLE}}, \pi^x_{\mathsf{proofs}}, \pi^x_{\mathsf{OLE}}, \mathsf{tag}^x, T_x, \overline{\mathsf{open}}^x\right), & k = 0; \\ \left(\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}^y_{\mathsf{OLE}}, \pi^y_{\mathsf{proofs}}, \pi^y_{\mathsf{OLE}}, \mathsf{tag}^y, T_y, \overline{\mathsf{open}}^y\right), & k = 1. \end{cases}$$
  3. Correlated randomness
$$r_{\mathsf{cor}} = \begin{cases} r_{\mathsf{sen}} = (r_{\mathsf{sen},1}, \ldots, r_{\mathsf{sen},\mathsf{m}}), & k = 0; \\ r_{\mathsf{rec}} = (r_{\mathsf{rec},1}, \ldots, r_{\mathsf{rec},\mathsf{m}}), & k = 1. \end{cases}$$
  4. An optional parameter π_{OLE}, which is the message output by the inner protocol. If $k = 0$, then we do not need this parameter. If $k = 1$, then $\pi_{\mathsf{OLE}} = \pi^x_{\mathsf{OLE}} = (\pi^1_{\mathsf{OLE},1}, \ldots, \pi^m_{\mathsf{OLE},1})$.
- **Execution:**
  - If $k = 0$, then the sender does:
    1. It checks if $H_T(\overline{\mathsf{com}}^x, \overline{\mathsf{com}}^U, \overline{\mathsf{com}}^x_{\mathsf{OLE}}, \pi^x_{\mathsf{proofs}}, \pi^x_{\mathsf{OLE}}, \mathsf{tag}^x)$ equals received T_x.
    2. For each $j \in T_x$,
       (a) It checks if $\mathsf{Open}(\mathsf{com}^x_j)$ successfully opens $\hat{x}_j$, $\mathsf{Open}(\mathsf{com}^U_j)$ successfully opens $U[j]$, and $\mathsf{Open}(\mathsf{com}^x_{\mathsf{OLE},\mathsf{j}})$ successfully opens $r_{\mathsf{rec},\mathsf{j}}$.
       (b) It checks if $(r_{\mathsf{sen},\mathsf{j}}, r_{\mathsf{rec},\mathsf{j}})$ is a valid pair of correlated randomness. For random bit-OT correlations, it checks if $(S^0, S^1) \in \{0,1\}^2$ is consistent with $(C, R = S^C)$. For random OLE correlations, it checks if $y_r = a_r x_r + b_r$.
       (c) It simulates the inner protocol $\Pi_{\mathsf{OLE},1}(\hat{x}_j, r_{\mathsf{rec},\mathsf{j}})$ and checks if the output is equal to the $\pi^j_{\mathsf{OLE},1}$ received.
       (d) It checks if $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2} \odot \hat{x}_{j,3}$.

    If any of the checks above fails, it outputs 0. Else, it outputs 1.
  - If $k = 1$, then the receiver does:
    1. It checks if $H_T(\overline{\mathsf{com}}^y, \overline{\mathsf{com}}^V, \overline{\mathsf{com}}^{V^0}, \overline{\mathsf{com}}^y_{\mathsf{OLE}}, \pi^y_{\mathsf{proofs}}, \pi^y_{\mathsf{OLE}}, \mathsf{tag}^y)$ equals received T_y.
    2. For each $j \in T_y$,
       (a) It checks if the openings $\mathsf{Open}(\mathsf{com}^y_j)$ successfully opens $\hat{y}_j$, $\mathsf{Open}(\mathsf{com}^V_j)$ successfully opens $V[j]$, $\mathsf{Open}(\mathsf{com}^{V^0}_j)$ successfully opens $V^0[j]$, and $\mathsf{Open}(\mathsf{com}^y_{\mathsf{OLE},\mathsf{j}})$ successfully opens $r_{\mathsf{sen},\mathsf{j}}$.
       (b) It checks if $(r_{\mathsf{sen},\mathsf{j}}, r_{\mathsf{rec},\mathsf{j}})$ is a valid pair of correlated randomness.
       (c) It simulates the inner protocol $\Pi_{\mathsf{OLE},2}(\pi^j_{\mathsf{OLE},1}, \hat{y}_j, \hat{0}_j, r_{\mathsf{sen},\mathsf{j}})$ and checks if the output is equal to the $\pi^j_{\mathsf{OLE},2}$ received.
       (d) It checks if $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2} \odot \hat{y}_{j,3}$.

    If any of the checks above fails, it outputs 0. Else, it outputs 1.

Figure 15: Description of $\mathsf{checkWatchlist}$

8.1 Free-XOR

Our protocol supports the Free-XOR optimization introduced in [KS08] assuming a random oracle. Essentially, the sender samples a global offset $\Delta \in \{0,1\}^\lambda \setminus \{0^\lambda\}$ uniformly at random. For each circuit input wire and each gate output wire of AND gates, denote the wire by w, and the sender samples $\mathtt{lab}_{w,0} \in \{0,1\}^\lambda$ and computes $\mathtt{lab}_{w,1} = \mathtt{lab}_{w,0} \oplus \Delta$. For an XOR gate represented by tuple (i,j,k), the labels for the output wire k are computed as $\mathtt{lab}_{k,0} = \mathtt{lab}_{i,0} \oplus \mathtt{lab}_{j,0}$ and $\mathtt{lab}_{k,1} = \mathtt{lab}_{k,0} \oplus \Delta$. This optimization can be directly applied in our protocol. Note that without the optimization, the sender shares the labels as $(\mathtt{lab}_{w,0}, (\mathtt{lab}_{w,0} \oplus \mathtt{lab}_{w,1}))$ for affected wires. Now, the sender shares the labels as $(\mathtt{lab}_{w,0}, \Delta)$.

The only additional communication cost we need to pay is that we need the sender to use the Ligero proof for linear constraints $(P,V)^{\mathcal{H}}_{\mathsf{lin}}$ to prove that Δ has the same value across all wires affected by the Free-XOR optimization. Note that Free-XOR also requires that the encryption of the garbled entries come from correlation-robust hash functions, which we get for free from random oracles.

8.2 Concrete Overhead of Our Protocol

We start with estimating the concrete overhead of our protocol over the semi-honest Yao's garbled circuit. Recall equation (1) in BMR garbling process: for each $\alpha, \beta \in \{0,1\}$, we compute the gate output bit by

$$z_{\alpha,\beta} = g\left((r_i \oplus s_i) \oplus \alpha, (r_j \oplus s_j) \oplus \beta\right) \oplus (r_k \oplus s_k).$$

Given the Free-XOR optimization above, we only consider AND gates, and plug in values of α, β and open the arithmetic computation, we have

$$\begin{aligned}
z_{0,0} &= r_i r_j + r_i s_j + s_i r_j + s_i s_j + r_k + s_k;\\
z_{0,1} &= r_i r_j + r_i s_j + r_i + s_i r_j + s_i s_j + s_i + r_k + s_k;\\
z_{1,0} &= r_i r_j + r_i s_j + s_i r_j + s_i s_j + r_j + s_j + r_k + s_k;\\
z_{1,1} &= r_i r_j + r_i s_j + r_i + s_i r_j + s_i s_j + s_i + r_j + s_j + 1 + r_k + s_k.
\end{aligned}$$

Therefore, the polynomials that compute $z_{0,0}, z_{0,1}, z_{1,0}, z_{1,1}$ contain $6, 8, 8, 11$ monomials respectively. Following the garbling process, then computing the gate output label $\mathtt{lab}_{k,z_{\alpha,\beta}}$ requires polynomials with $7, 9, 9, 12$ monomials respectively. Similarly, computing the MAC $\mathtt{mac}_{g,\alpha,\beta}$ requires polynomials with $7, 9, 9, 12$ monomials respectively. Thus, to compute $M_{\alpha,\beta}$ as in equation (3), it takes $14, 18, 18, 24$ monomials respectively. In the end, in order to do the PRF-based encryption, one extra monomial is added to the computation of $\mathtt{lab}_{k,z_{\alpha,\beta}}$ and one extra monomial is added to the computation of $\mathtt{mac}_{g,\alpha,\beta}$. Hence, to compute the garbled entry $g_{0,0}, g_{0,1}, g_{1,0}, g_{1,1}$, it takes $16, 20, 20, 26$ monomials respectively. Putting these monomials in a vector, the vector will have $16 + 20 + 20 + 26 = 82$ entries. Taking $|C|$ to be the number of AND gates in the circuit, the long vector $\overline{p}$ that contains all the information to share has $82 \cdot |C|$ field elements.

Recall that we use a packed secret sharing of packing size ℓ, so the number of vectors to share is $82 \cdot |C|/\ell$.

Now consider the inner protocol, which essentially evaluates OLEs. Assuming OLE correlations, the protocol $\Pi_{\mathsf{singleOLE}}$ exchanges 3 field elements to compute a single standard OLE. Since we have m servers, each running an independent instance of the inner protocol, the total communication

cost in bits is

$$m \cdot 3 \cdot 82 \cdot |C|/\ell \cdot \lambda = 246 \cdot m/\ell \cdot |C|\lambda.$$

In the final compiled protocol, the communication cost is dominated by the communication cost of the inner protocol. The key observation is that the proofs and the watchlist only opens a subset of the servers, so the communication cost incurred by proofs and the watchlist will have a term $|T|/\ell$, where $|T|$ is the size of the subset of servers opened. By choosing proper parameters, $|T|/\ell$ can be made insignificant when compared to the communication cost of the inner protocol.

Note that in Φ_{circ}, the vectors $\hat{x}_{j,1}, \hat{x}_{j,2}, \hat{x}_{j,3}$ (resp. $\hat{y}_{j,1}, \hat{y}_{j,2}, \hat{y}_{j,3}$) are shared using a degree-$(t+\ell)$ polynomial. Thus, the vectors $\hat{x}_j = \hat{x}_{j,1} \odot \hat{x}_{j,2} \odot \hat{x}_{j,3}$ and $\hat{y}_j = \hat{y}_{j,1} \odot \hat{y}_{j,2} \odot \hat{y}_{j,3}$ will be held by degree-$(3t+3\ell)$ polynomials, and the final result will be held by a degree-$(6t+6\ell)$ polynomial. This requires the number of servers $m \geq 8t + 6\ell + 1$, where t, ℓ are constant fraction of m. Therefore, if we pick t to be an insignificant small constant fraction of m, the ratio m/ℓ approaches 6. Thus, the total communication cost of our protocol is roughly $6 \times 246|C|\lambda = 1476|C|\lambda$. Compared with the semi-honest Yao's garbled circuit, which exchanges 4 field elements per garbled gate, our overhead is $1476/4 = 369$.

8.3 Further Improvement of the Concrete Overhead

Following the estimation above, we can observe that the communication cost is essentially

$$m/\ell \cdot 3 \cdot |\text{BMR monomials}| \cdot |C|\lambda. \tag{12}$$

The m/ℓ ratio is non-trivial to decrease below 6 due to the nature of the decomposition process of our protocol. The 3 comes from the number of elements exchanged in an OLE protocol, which is also hard to improve. Therefore, in order to reduce the concrete cost, we need to reduce the number of monomials incurred from the BMR garbling process.

Let us revisit equation (1) and take g as an AND gate:

$$\begin{aligned} z_{\alpha,\beta} &= g\left((r_i \oplus s_i) \oplus \alpha, (r_j \oplus s_j) \oplus \beta\right) \oplus (r_k \oplus s_k) \\ &= (r_i + s_i + \alpha)(r_j + s_j + \beta) + r_k + s_k \\ &= (r_i r_j + \beta r_i + \alpha r_j + r_k) + (s_i s_j + \beta s_i + \alpha s_j + s_k) + r_i s_j + s_i r_j + \alpha\beta \end{aligned}$$

Naively expanding the above expression will create 11 monomials, but we can group the terms according to whether the variables in the monomial comes from the sender or from the receiver. Following the garbling process, we then compute the MAC code

$$\begin{aligned} \mathtt{mac}_{g,\alpha,\beta} = \mathtt{key}_g(r_i r_j + \beta r_i + \alpha r_j + r_k) + \mathtt{key}_g(s_i s_j + \beta s_i + \alpha s_j + s_k) \\ + \mathtt{key}_g r_i s_j + \mathtt{key}_g s_i r_j + \alpha\beta \mathtt{key}_g + b_g \end{aligned}$$

and the gate output label

$$\begin{aligned} \mathtt{lab}_{k,z_{\alpha,\beta}} = \mathtt{lab}_{k,0} + (r_i r_j + \beta r_i + \alpha r_j + r_k)\Delta + (s_i s_j + \beta s_i + \alpha s_j + s_k)\Delta \\ + r_i s_j \Delta + s_i r_j \Delta + \alpha\beta\Delta \end{aligned}$$

where $\Delta = (\mathtt{lab}_{k,0} \oplus \mathtt{lab}_{k,1})$. In the end, we compute the PRF-based encryption

$$g_{\alpha,\beta} = (\mathtt{mac}_{g,\alpha,\beta} + \mathcal{F}_1, \mathtt{lab}_{k,z_{\alpha,\beta}} + \mathcal{F}_2)$$

where $\mathcal{F}_1$ (resp. $\mathcal{F}_2$) is the first (resp. second) λ bits of $\mathcal{F}(\mathtt{lab}_{i,\alpha}, g, \beta) + \mathcal{F}(\mathtt{lab}_{j,\beta}, g, \alpha)$, which was

pre-computed by the sender. Thus, our goal is to reduce the number of monomials in $g_{\alpha,\beta}$. This is doable with a minor tweak in the inner protocol.

First, we group the monomials of $\mathtt{mac}_{g,\alpha,\beta} + \mathcal{F}_1$ and $\mathtt{lab}_{k,z_{\alpha,\beta}} + \mathcal{F}_2$ by the party that provides it. There will be some cross-terms. We have

$$\begin{aligned}\mathtt{mac}_{g,\alpha,\beta} + \mathcal{F}_1 = \mathtt{key}_g(r_i r_j + \beta r_i + \alpha r_j + r_k) + \mathtt{key}_g(s_i s_j + \beta s_i + \alpha s_j + s_k + \alpha\beta) \\ + \mathtt{key}_g r_i s_j + \mathtt{key}_g s_i r_j + b_g + \mathcal{F}_1,\end{aligned}$$

which consists of four categories:

- $\mathtt{key}_g(r_i r_j + \beta r_i + \alpha r_j + r_k) + b_g$ all come from the receiver.
- $\mathtt{key}_g(s_i s_j + \beta s_i + \alpha s_j + s_k + \alpha\beta)$ has one variable $\mathtt{key}_g$ from receiver, and everything else from the sender.
- $\mathtt{key}_g r_i s_j$ and $\mathtt{key}_g s_i r_j$ have two variables from the receiver, and one variable from the sender.
- $\mathcal{F}_1$ comes entirely from the sender.

Therefore, in order to represent $\mathtt{mac}_{g,\alpha,\beta} + \mathcal{F}_1$, let

$$v_{\mathsf{mac}} = \begin{pmatrix} \mathtt{key}_g(r_i r_j + \beta r_i + \alpha r_j + r_k) + b_g \\ \mathtt{key}_g(s_i s_j + \beta s_i + \alpha s_j + s_k + \alpha\beta) \\ \mathtt{key}_g r_i s_j \\ \mathtt{key}_g s_i r_j \\ \mathcal{F}_1 \end{pmatrix}$$

be a vector of 5 elements. We can decompose v_{mac} as $v^x_{\mathsf{mac}} \odot v^y_{\mathsf{mac}}$, where

$$v^x_{\mathsf{mac}} = \begin{pmatrix} \mathtt{key}_g(r_i r_j + \beta r_i + \alpha r_j + r_k) + b_g \\ \mathtt{key}_g \\ \mathtt{key}_g r_i \\ \mathtt{key}_g r_j \\ 1 \end{pmatrix},\ v^y_{\mathsf{mac}} = \begin{pmatrix} 1 \\ (s_i s_j + \beta s_i + \alpha s_j + s_k + \alpha\beta) \\ s_j \\ s_i \\ \mathcal{F}_1 \end{pmatrix}$$

The above vectors can be further decomposed to degree-1 vectors. For example, for v^x_{mac}, the receiver can share below vectors:

$$\begin{aligned} v^x_1 &= \begin{pmatrix} \mathtt{key}_g & \mathtt{key}_g & \mathtt{key}_g & \mathtt{key}_g & 1 \end{pmatrix} \\ v^x_2 &= \begin{pmatrix} b_g & 0 & 0 & 0 & 0 \end{pmatrix} \\ v^x_3 &= \begin{pmatrix} r_k & 1 & r_i & r_j & 1 \end{pmatrix} \\ v^x_4 &= \begin{pmatrix} r_i & 0 & 0 & 0 & 0 \end{pmatrix} \\ v^x_5 &= \begin{pmatrix} r_j & 0 & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

Observe that v^x_{mac} can be computed by $(v^x_4 \odot v^x_5 + \beta v^x_4 + \alpha v^x_5 + v^x_3) \odot v^x_1 + v^x_2$. Note that this still has degree $3(t+\ell)$ so it does not change the m/ℓ ratio. Informally, we call $v^x_1, \ldots, v^x_5$ *explanations* of the vector v^x_{mac}. Similarly, the sender generates explanations for the vector v^y_{mac}. Note that these explanations are not actually sent to the servers in the compiled final protocol. Instead, they are committed and proved by linear constraints. The execution of the adapted inner protocol will be checked by the watchlist. It increases the communication cost from the proofs and the compiler part. However, the communication cost is still dominated by the communication cost of the inner protocol. To summarize, we reduced the number of monomials to compute the polynomial

$\mathtt{mac}_{g,\alpha,\beta} + \mathcal{F}_1$ for one garbled entry to 5.

Similarly, consider the polynomial $\mathtt{lab}_{k,z_{\alpha,\beta}} + \mathcal{F}_2$, which can be written as

$$\begin{aligned}\mathtt{lab}_{k,z_{\alpha,\beta}} + \mathcal{F}_2 = \mathtt{lab}_{k,0} &+ (r_i r_j + \beta r_i + \alpha r_j + r_k + \alpha\beta)\Delta \\ &+ (s_i s_j + \beta s_i + \alpha s_j + s_k)\Delta \\ &+ r_i s_j \Delta + s_i r_j \Delta + \mathcal{F}_2.\end{aligned}$$

Following the same grouping procedure, we represent $\mathtt{lab}_{k,z_{\alpha,\beta}} + \mathcal{F}_2$ by

$$v_{\mathsf{lab}} = \begin{pmatrix} (s_i s_j + \beta s_i + \alpha s_j + s_k)\Delta + \mathtt{lab}_{k,0} + \mathcal{F}_2 \\ (r_i r_j + \beta r_i + \alpha r_j + r_k + \alpha\beta)\Delta \\ r_i s_j \Delta \\ s_i r_j \Delta \end{pmatrix}.$$

Again, we can decompose it to v^x_{lab} and v^y_{lab} and further use the explanations to compute these vectors. In this way, we reduced the number of monomials to compute the polynomial $\mathtt{lab}_{k,z_{\alpha,\beta}} + \mathcal{F}_2$ to 4.

Therefore, after the above optimization, it takes $5 + 4 = 9$ monomials to compute each $g_{\alpha,\beta}$. Therefore, the total communication cost of the inner protocol over all m servers comes down to $m \cdot 3 \cdot (9 \times 4) \cdot |C|\lambda/\ell = 108|C|\lambda m/\ell$. Since m/ℓ approaches 6, the communication cost approaches $648|C|\lambda$, which has a $648/4 = 162$ overhead compared with semi-honest Yao's garbled circuits.

Acknowledgments. The authors thank the anonymous reviewers for their detailed comments and Swastik Kopparty for helpful discussions. Yuval Ishai was supported in part by ISF grants 2774/20 and 3527/24 and BSF grant 2022370. This work is not associated with Amazon. Akshayaram Srinivasan and Ziyang Jin are supported in part by the NSERC Discovery Grant RGPIN-2024-03928.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[BCG+20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

[BCG+20b] 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, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[BCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *61st FOCS*, pages 900–909, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[BGMN18] Alexander R. Block, Divya Gupta, Hemanta K. Maji, and Hai H. Nguyen. Secure computation using leaky correlations (asymptotically optimal constructions). In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 36–65, Panaji, India, November 11–14, 2018. Springer, Cham, Switzerland.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[BS05] Eli Ben-Sasson and Madhu Sudan. Simple PCPs with poly-log rate and query complexity. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 266–275, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 395–426, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[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, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[CHHK25] Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. $\omega(1/\lambda)$-rate boolean garbling scheme from generic groups. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 489–521, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[CRX21] Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. Asymptotically-good arithmetic secret sharing over $\mathbb{Z}/p^{\ell}\mathbb{Z}$ with strong multiplication and its applications to efficient MPC. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 656–686, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 35–67, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[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, Santa Barbara, CA, USA, August 14–18, 2005. Springer Berlin Heidelberg, Germany.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520, Santa Barbara, CA, USA, August 20–24, 2006. Springer Berlin Heidelberg, Germany.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[EXY22] Daniel Escudero, Chaoping Xing, and Chen Yuan. More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via Galois rings. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 383–412, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710, Victoria, BC, Canada, May 4–6, 1992. ACM Press.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GLOS25] Vipul Goyal, Junru Li, Rafail Ostrovsky, and Yifan Song. Towards building scalable constant-round MPC from minimal assumptions via round collapsing. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 70–104, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

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

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[HIV17] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 3–39, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284, San Francisco, CA, USA, May 21–23, 2012. IEEE Computer Society Press.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany.

[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 16–20, 2021. Springer, Cham, Switzerland.

[IKSS22a] Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-optimal black-box protocol compilers. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 210–240, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[IKSS22b] Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-optimal black-box secure computation from two-round malicious OT. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 441–469, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland.

[IKSS23] Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Black-box reusable NISC with random oracles. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 68–97, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[ILL25] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 390–425, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[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, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.

[JRR25] Jake Januzelli, Mike Rosulek, and Lawrence Roy. Lower bounds for garbled circuits from shannon-type information inequalities. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part IV*, volume 16003 of *Lecture Notes in Computer Science*, pages 589–618. Springer, 2025.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M.

Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer Berlin Heidelberg, Germany.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78, Barcelona, Spain, May 20–24, 2007. Springer Berlin Heidelberg, Germany.

[LWYY25] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled circuits with 1 bit per gate. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 437–466, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473, New York, NY, USA, April 24–26, 2006. Springer Berlin Heidelberg, Germany.

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 426–458, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[NO16] Jesper Buus Nielsen and Claudio Orlandi. Cross and clean: Amortized garbled circuits with constant overhead. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 582–603, 2016.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337, Santa Barbara, CA, USA, August 17–21, 2003. Springer Berlin Heidelberg, Germany.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *26th ACM STOC*, pages 194–203, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction for algebraic block codes. U.S. Patent 4,633,470, December 1986. Issued December 30, 1986.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[XHX25] Fei Xu, Honggang Hu, and Changhong Xu. Bitwise garbling schemes - A model with $\frac{3}{2}\kappa$-bit lower bound of ciphertexts. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part IV*, volume 16003 of *Lecture Notes in Computer Science*, pages 459–488. Springer, 2025.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.