# Two Levels are Better than One: Dishonest Majority MPC with $\widetilde{O}(|C|)$ Total Communication

Alexander Bienstock[1] and Kevin Yeo[2]

[1] New York University
[2] Google and Columbia University

**Abstract.** *packing parameter* In recent years, there has been tremendous progress in improving the communication complexity of dishonest majority MPC. In the sub-optimal corruption threshold setting, where $t < (1 - \varepsilon) \cdot n$ for some constant $0 < \varepsilon \leq 1/2$, the recent works Sharing Transformation (Goyal *et al.*, CRYPTO'22) and SuperPack (Escudero *et al.*, EUROCRYPT'23) presented protocols with information-theoretic online phases achieving $O(1)$ communication per multiplication gate, across all parties. However, the former assumes that their offline phase is instantiated by a trusted party, while the latter instantiates their offline phase with $\Omega(n)$ communication per multiplication gate assuming oblivious linear evaluation (OLE) correlations.

In this work, we present a dishonest majority MPC protocol for $t < (1 - \varepsilon) \cdot n$ with $\widetilde{O}(1)$ total communication per multiplication gate across both the offline and online phases, or $\widetilde{O}(|C|)$ total communication for any arithmetic circuit $C$. To do so, we securely instantiate the offline phase of Sharing Transformation, assuming some OLE correlations. The major bottleneck in instantiating the offline phases of both Sharing Transformation and SuperPack is generating random packed beaver triples of the form $[\boldsymbol{a}], [\boldsymbol{b}], [\boldsymbol{c}]$, for random $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$, and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b} \in \mathbb{F}^k$, where $k = \Omega(n)$ is the *packing parameter*. We overcome this barrier by presenting a packed beaver triple protocol with $\widetilde{O}(n)$ total communication, or $\widetilde{O}(1)$ communication per underlying triple.

Our packed beaver triple protocol consists of two levels of randomness extraction. The first level uses a relaxation of super-invertible matrices that we introduce, called *weakly* super-invertible matrices, in which sub-matrices have sufficiently high (but not necessarily full) rank. This weakening enables matrix constructions with only $O(n)$ non-zero entries, which is a primary reason for the efficiency of our protocol. Our second level of extraction is based on the *triple extraction* protocol of (Choudhury and Patra, Trans. Inform. Theory '17).

## 1 Introduction

Secure Multi-Party Computation (MPC) is a widely studied area of cryptography [GMW87, Yao82, CCD88, BGW88]. In MPC, there are $n$ parties with respective private inputs $x_1, \ldots, x_n$, and some functionality $\mathcal{F}$. The parties wish to interact with each other to compute and output $\mathcal{F}(x_1, \ldots, x_n)$ in such a way that their inputs still remain hidden, beyond what can be inferred by the output. In fact, this should still hold even if there is an adversary controlling $t$ out of $n$ of the parties, who we call *corrupted*. The adversary sees all of the messages that the other parties, who we call *honest*, send these corrupted parties. Adversaries can either be *static* or *adaptive*. In the former case, the adversary must choose which parties to corrupt before the protocol begins. In the latter case, the adversary chooses which parties to corrupt during the protocol, possibly based on the observed messages.

In this work, we focus on the so-called *dishonest majority* setting in which the number of corrupted parties is $t \geq n/2$. In this setting, it is known that protocols require computational assumptions to be proved secure [RB89], unlike in the case where $t < n/2$ [BGW88, CCD88, RB89]. However, many dishonest majority protocols operate in the *offline-online* model, in which there is first an offline phase that is typically proved computationally-secure and then an online phase that can be proved information-theoretically secure [Bea92]. This model has the benefit that expensive cryptographic operations can be performed in the offline phase (at any point before the functionality and inputs are determined), meaning that the online phase is usually quite efficient. Typically, the offline phase provides the parties with some correlated randomness, which they consume in the online phase. This is the model which we focus on in this work.

In this setting, the main goal has been to reduce the communication complexity of protocols. The seminal works of BeDOZa [BDOZ11] and SPDZ [DPSZ12, DKL$^+$13] achieve linear communication complexity in the number of parties per multiplication gate, in both the offline phase and online phase. Achieving this efficiency in the offline phase required heavy cryptographic tools such as fully-homomorphic encryption (FHE). However, FHE is prohibitively expensive and many prior works explicitly dismiss it as such [DPSZ12, EGP$^+$23]. Furthermore, FHE usage presents significant barriers to adaptive security (see [KTZ13, CsW19]). Without FHE, these offline phases were instantiated with quadratic communication complexity in the number of parties per gate, with the same online phase that achieves linear communication per gate.

The offline phases of these protocols produce simple correlated randomness, in the form of oblivious linear evaluation (OLE) [IPS09, AIK11], for every pair of parties. An OLE correlation between parties $P_A$ and $P_B$ samples consists of random $x, u, v$ such that $P_A$ receives $x$ and $v$, and $P_B$ receives $u$ and $w = u \cdot x - v$. More recent works have shown that the OLE correlations needed by these protocols' offline phases can be instantiated with communication complexity *sub-linear* in the *circuit* size [RS22], using so-called pseudo-correlation generators [BCG$^+$19b, BCG$^+$20, BCGI18, WYKW21, BCG$^+$19a].[3] All of the above works consider an *optimal* corruption threshold, $t = n - 1$, where all but one parties could be corrupted.

The tremendous recent works Sharing Transformation [GPS22] and SuperPack [EGP$^+$23] instead study a *sub-optimal* corruption threshold, in which $t < (1 - \varepsilon) \cdot n$, for some constant $0 < \varepsilon \leq 1/2$. Here, they are able to obtain

[3] Note that, typically, the circuit size is the dominating term.

online phases with communication complexity $O(1)$ per gate.[4] Both of these protocols at their core take advantage of *packed* Shamir secret sharing [FY92], which allows parties to secret share many (proportional to $n$) values at once. However, the Sharing Transformation work does not instantiate their offline phase, which provides the parties with highly non-trivial correlated randomness. Any currently-available generic protocol would unfortunately have *linear* communication in the number of parties per gate to instantiate this correlated randomness. This is much worse than the $O(1)$ per gate during the online phase. SuperPack does instantiate their offline phase, which still assumes some some simple OLE correlations, and requires *linear* communication in the number of parties per gate, even assuming this correlated randomness. Again, this is much worse than the $O(1)$ per gate during the online phase.

If one only desires static security, then there is a simple protocol with sub-linear total communication in the number of parties in the $t < (1-\varepsilon)\cdot n$ setting [HIK07]. The parties first randomly select a sub-committee of parties of size $\kappa/\varepsilon$, where $\kappa$ is the *computational* security parameter. It can be seen that with $1 - 2^{-\Omega(\kappa)}$ probability, at least one member of this sub-committee will not be corrupted by the static adversary. Therefore, the parties can use this sub-committee to carry out the rest of the MPC for them, using an efficient protocol for $t = \kappa/\varepsilon - 1$ corruptions, such as SPDZ or BeDOZa. In this case, the communication complexity per gate will only depend on $\kappa/\varepsilon = \widetilde{O}(1)$, the number of parties in the committee. Indeed, it will be independent of the total number of parties, $n$. However, this simple protocol is trivially broken by an adaptive adversary who gets to see the members of the sub-committee, and then corrupt all of them. If one were allowed to use FHE, then the the offline phases of the Sharing Transformation and SuperPack works could also be instantiated with $O(1)$ communication per gate. However, again, FHE is a heavy cryptographic tool that is very computationally expensive and faces significant barriers in being proved adaptively-secure [KTZ13, CsW19].

This begs the following question, which we address in our work:

*Can we achieve adaptive dishonest majority MPC with $\widetilde{O}(1)$ communication per multiplication gate in both the offline and online phases?*

To understand the challenges with this problem, we outline the major obstacles from prior works and our solution to each of them.

**Packed Beaver Multiplication Triples.** A common tool used in dishonest majority MPC is beaver multiplication [Bea92]. In the $t < (1-\varepsilon)\cdot n$ setting, both of the above works [GPS22, EGP$^+$23] utilize this technique through packed Shamir secret sharing [FY92]. For $\Theta(n)$-dimensional *vectors* $\boldsymbol{x}$ and $\boldsymbol{y}$, the parties wish to compute a packed Shamir sharing of the vector of multiplications $\boldsymbol{x} * \boldsymbol{y}$ on input packed Shamir sharings $[\boldsymbol{x}]$ and $[\boldsymbol{y}]$, where $*$ denotes component-wise multiplication. They do so using packed Shamir sharings $[\boldsymbol{a}]$, $[\boldsymbol{b}]$, and $[\boldsymbol{c}]$, such that $\boldsymbol{a}$ and $\boldsymbol{b}$ are fully random $\Theta(n)$ dimensional vectors, and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. These so-called

[4] The latter is more concretely efficient.

packed beaver triples allow the parties to compute $[\boldsymbol{x} * \boldsymbol{y}]$ with communication linear in $n$. Since this is for $\Theta(n)$ gates at once, the communication per gate is therefore $O(1)$. Indeed, the protocols of [GPS22, EGP$^+$23] require that these packed beaver triples $[\boldsymbol{a}]$, $[\boldsymbol{b}]$, $[\boldsymbol{c}]$ be generated during the offline phase. This turns out to be the main reason that they can only achieve linear communication per gate in the offline phase. In particular, generating these packed triples alone requires $\Omega(n^2)$ communication per packed triple, and thus $\Omega(n)$ per underlying triple, using known techniques. In our work, we show how to resolve this issue by presenting a protocol which achieves $\widetilde{O}(n)$ communication per packed triple.

**(Weakly) Super-Invertible Matrices.** Another common tool used in efficient MPC protocols are super-invertible $m \times n$ matrices $\boldsymbol{M}$, where $m \leq n$. These matrices satisfy that for all column subsets $C \subseteq [n]$ with $|C| = m$, the sub-matrix $\boldsymbol{M}^C$ corresponding to the columns with indices in $C$ is invertible. These matrices are typically used to efficiently sample secret sharings of random values [DN07]. In the $t < (1-\varepsilon) \cdot n$ setting, both [GPS22, EGP$^+$23] utilize super-invertible matrices to generate packed Shamir sharings $[\boldsymbol{r}]$ of random vectors with $O(1)$ communication per individual random value. The usual instantiation of super-invertible matrices is the transpose of a Vandermonde matrix [DN07]. Such matrices are *dense*, meaning that almost all entries are non-zero. In our work, we introduce a relaxation of super-invertible matrices, which we call *weakly* super-invertible matrices, that will actually allow us to generate packed beaver triples with $\widetilde{O}(n)$ communication per packed triple. The main reason is that we are able to construct weakly super-invertible matrices that are *sparse*; i.e., most entries are zero. Weak super-invertibility requires that sub-matrices consisting of sufficiently many columns have large (but not necessarily full) rank. We will later show that this property is sufficient for efficiently generating packed beaver triples.

## 1.1 Our Contributions

**Packed Beaver Triples with $\widetilde{O}(1)$ Communication per Triple.** The first result of our paper is providing a protocol that generates $\Theta(n)$ random packed beaver triples with $\widetilde{O}(n)$ communication per packed triple, or $\widetilde{O}(1)$ communication per underlying triple. Our protocol assumes $O(n)$ OLE correlations for each of $\widetilde{O}(n)$ pairs of parties. Note, this is not all $O(n^2)$ pairs of parties.

**Theorem 1 (Informal).** *Let $\lambda$ be the statistical security parameter. There exists an information-theoretically secure protocol in the OLE-model which securely generates $\Theta(n)$ packed beaver triples in the presence of a malicious, adaptive adversary controlling up to $t < (1-\varepsilon) \cdot n$ parties. The cost of the protocol is $O(n)$ OLE correlations between each of $O(n \cdot \lambda^2)$ pairs of parties and $O(n^2 \cdot \lambda)$ elements of communication, which is $\widetilde{O}(n)$ per packed triple and $\widetilde{O}(1)$ per underlying triple.*

*Remark 1.* By using any generic 2PC protocol secure in the presence of a malicious, adaptive adversary (e.g., [CLOS02]), we can instantiate each OLE correlation with $\texttt{poly}(\kappa, \log|\mathbb{F}|) = \texttt{poly}(\kappa)$ communication complexity. Therefore, even when counting the communication from the underlying OLE instantiations, the communication of our protocol is still $\widetilde{O}(1)$ per underlying triple.

**$\widetilde{O}(|C|)$ Communication MPC for $t < (1-\varepsilon)\cdot n$.** We use the above result for the main contribution of our paper, which is a protocol that instantiates the offline phase of [GPS22] with $\widetilde{O}(1)$ communication per multiplication gate. Using their online phase, this implies an MPC protocol in the $t < (1-\varepsilon)\cdot n$ setting with $\widetilde{O}(1)$ *total* communication per multiplication gate.

**Theorem 2 (Informal).** *Let $\lambda$ be the statistical security parameter. For an arithmetic circuit $C$ over a finite field $\mathbb{F}$, there exists an information-theoretically secure MPC protocol in the OLE-model which securely computes the arithmetic circuit $C$ in the presence of a malicious, adaptive adversary controlling up to $t < (1-\varepsilon)\cdot n$ parties. The cost of the protocol is $O(|C|/n)$ OLE correlations between each of $O(n\cdot\lambda^2)$ pairs of parties and $O(|C|\cdot\lambda+n^2\cdot\lambda)$ elements of communication. This is $\widetilde{O}(1)$ per gate when the circuit is sufficiently large, $|C| = \widetilde{\Omega}(n^2)$.*

*Remark 2.* Following from Remark 1, even when counting the communication from the underlying OLE instantiations, the communication of our protocol is still $\widetilde{O}(1)$ per gate.

*Remark 3.* Our techniques are insufficient for realizing the offline phase of [EGP+23] with $\widetilde{O}(1)$ communication per gate. Intuitively, this is because it requires *individual* random values to be shared in *multiple* packed Shamir secret sharings containing different values. Thus, the typical technique of using super-invertible matrices to efficiently generate several *independent* random packed Shamir secret sharings at once fails. See Section 2 for more details.

**Weakly Super-Invertible Matrices.** As mentioned above, the main technique that we develop is generating packed beaver triples with $\widetilde{O}(n)$ communication per triple. A crucial part of this technique is a weakening of super-invertible matrices that we develop, called *weakly super-invertible matrices* requiring that sub-matrices have sufficiently high (but not full) rank. We say a $m \times n$ matrix is weakly super-invertible if every $m \times \Theta(m)$ sub-matrix has rank at least $\gamma \cdot m$ for some constant $\gamma > 1/2$. This is a weaker property than standard super-invertibility requiring full rank. Nevertheless, we later show that this property is still sufficient for our MPC applications. We are able to construct weakly super-invertible matrices that are sparse with only $O(n)$ non-zero entries.

**Theorem 3 (Informal).** *For any constants $1/2 < \gamma < \eta < 1$, any $n = \Theta(m)$ and any finite field $\mathbb{F}$, there exists a matrix generation algorithm that outputs random $m \times n$ matrices satisfying the following except with probability $2^{-\lambda}$:*

1. *Every sub-matrix of dimension $m \times (\eta \cdot m)$ will have rank at least $\gamma \cdot m$.*
2. *Every row will have at most $O(\lambda + \log m)$ non-zero entries.*

*Additionally, the whole matrix* always *has at most $O(n \cdot \log |\mathbb{F}|)$ non-zero entries.*

### 1.2 Related Works

**Dishonest Majority MPC in the offline-online model.** There are many dishonest majority MPC protocols that work in the offline-online mode. Most notably, this includes BeDOZa [BDOZ11], SPDZ [DPSZ12, DKL$^+$13], MASCOT [KOS16], Overdrive [KPR18], TopGear [BCS19], and many more.

**Honest Majority MPC with Sub-Optimal Threshold.** In the setting of honest majority information-theoretic MPC with sub-optimal threshold, where the number of corrupted parties is $t < (1/2 - \varepsilon) \cdot n$, there has been a fruitful line of works achieving sub-linear communication per multiplication gate such as [FY92, DIK10, GPS21, GIP15, GIOZ17, BGJK21, GSY21]. Indeed, the work of [GPS21] shows how to get $O(1)$ communication per multiplication gate in this setting.

**Invertible Matrices.** Invertible matrices have been an important tool in the construction of MPC. Hirt and Nielsen [HN06] introduced super-invertible matrices and Beerliová-Trubíniová and Hirt [BTH08] extended this to hyper-invertibility. These matrices have been used in MPC in several ways including for randomness extraction and error correction [DN07, BTH08]. To our knowledge, the main instantiation for either matrix definition is Vandermonde matrices.

## 2 Technical Overview

The main goal of this paper is to instantiate the offline phase of the $t < (1-\varepsilon)\cdot n$ MPC protocol of [GPS22] with $\widetilde{O}(1)$ communication per gate, which itself has an online phase with communication $O(1)$ per multiplication gate. A crucial tool that this online phase uses to achieve $O(1)$ communication is *packed Shamir secret sharing* [FY92], which allows to pack $\Theta(n)$ values in a single secret sharing, for a total cost of $O(n)$.

Indeed, the main object and primary efficiency bottleneck that the online phase of [GPS22] requires is that of *packed beaver triples.* This technical overview is dedicated to describing our protocol for generating packed beaver triples with $\widetilde{O}(1)$ communication per underlying triple in the OLE-model, an $\widetilde{\Omega}(n)$ multiplicative factor improvement over the best previous [EGP$^+$23].

**Packed Shamir Secret Sharing.** Packed Shamir secret sharing, introduced by [FY92] is a generalization of standard Shamir secret sharing [Sha79]. With packed Shamir secret sharing, we can pack a vector $\boldsymbol{x} \in \mathbb{F}^k$ of $k$ secrets into one degree-$d$ sharing, $[\boldsymbol{x}]_d$. Reconstructing a degree-$d$ packed Shamir sharing requires $d+1$ shares and can be done using Lagrange interpolation. Moreover, for a random degree-$d$ packed Shamir sharing of $\boldsymbol{x}$, any $d-k+1$ shares are independent of the secret $\boldsymbol{x}$. Packed Shamir sharings have the following properties:

- Linearity: For any $n > d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x}+\boldsymbol{y}]_d = [\boldsymbol{x}]_d + [\boldsymbol{y}]_d$.
- Multiplicative: For any $d_1, d_2 \geq k-1$ such that $d_1 + d_2 < n$, and for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x}*\boldsymbol{y}]_{d_1+d_2} = [\boldsymbol{x}]_{d_1} * [\boldsymbol{y}]_{d_2}$, where the $*$ operation is component-wise multiplication.

Note that the second property implies that for all $k-1 \leq d \leq n-k$, a degree-$d$ packed Shamir secret sharing is *multiplication-friendly*. What we mean by this is that for all $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{F}^k$, all parties can locally compute $[\boldsymbol{c} * \boldsymbol{x}]_{d+k-1}$ from $[\boldsymbol{x}]_d$ and public vector $\boldsymbol{c}$. To do this, all parties just locally transform $\boldsymbol{c}$ to some canonical degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{c}]_{k-1}$ and then use the $*$ operation.

Recall that $t$ is the number of corrupted parties and that a degree-$d$ packed Shamir secret sharing is private against an adversary that holds $d-k+1$ shares. To ensure that the Shamir secret sharing is both private and multiplication-friendly, we choose $k$ such that $t \leq d-k+1$ and $d \leq n-k$. When $d = n-k$ and $k = (n-t+1)/2$, both requirements hold and $k$ is maximal.

## 2.1 Packed Beaver Triples with $\widetilde{O}(1)$ Communication Per Triple

A packed beaver triple consists of the sharings $[\boldsymbol{a}]_{n-k}$, $[\boldsymbol{b}]_{n-k}$, and $[\boldsymbol{c}]_{n-k}$ such that $\boldsymbol{a}$ and $\boldsymbol{b}$ are fully random, while $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. We first show overview how to generate such packed beaver triples with a semi-honest adversary. Generating such packed beaver triples with malicious security will follow from standard techniques, as we will show later.

**Attempt 1: Packed Beaver Triples using Super-Invertible Matrices.** First, we will attempt to generate such packed beaver triples using *super-invertible* matrices and OLE correlations. Although we will fail in this attempt, it will provide intuition for our ultimate construction.

Recall that a $m \times n$ super-invertible matrix $\boldsymbol{M}$ satisfies that for all $C \subseteq [n]$ such that $|C| = m$, the sub-matrix $\boldsymbol{M}^C$, corresponding to the columns of $\boldsymbol{M}$ with indices in $C$, is invertible. Super-invertible matrices are commonly used to generate several random packed Shamir sharings $[\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_{n-k}$ with $O(n)$ amortized communication per sharing, and thus $O(1)$ communication per underlying value. To do so, each party $P_i$ begins by sampling their own random sharing $[\boldsymbol{u}_i]_{n-k}$ and distributing to the other parties their shares. Then, using a $(n-t) \times n$ super-invertible matrix $\boldsymbol{M}$, the parties simply compute $([\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_{n-k})^\intercal \leftarrow \boldsymbol{M} \cdot ([\boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{u}_n]_{n-k})^\intercal$. Intuitively, for any set of honest parties of size $n-t$, $H = \{h_1, \ldots, h_{n-t}\} \subseteq \mathsf{Hon}$, letting $C = \{c_1, \ldots, c_t\} = [n] \setminus \mathsf{Hon}$, we can write

$$([\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_{n-k})^\intercal = \boldsymbol{M}^H \cdot ([\boldsymbol{u}_{h_1}]_{n-k}, \ldots, [\boldsymbol{u}_{h_{n-t}}]_{n-t})^\intercal + \boldsymbol{M}^C \cdot ([\boldsymbol{u}_{c_1}]_{n-k}, \ldots, [\boldsymbol{u}_{c_t}]_{n-k})^\intercal.$$

Since $\boldsymbol{M}$ is super-invertible, $\boldsymbol{M}^H$ must be invertible, and thus, given any $([\boldsymbol{u}_{c_1}]_{n-k}, \ldots, [\boldsymbol{u}_{c_t}]_{n-k})$, $([\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_{n-k})$ and $([\boldsymbol{u}_{h_1}]_{n-k}, \ldots, [\boldsymbol{u}_{h_{n-t}}]_{n-t})$ are distributed equivalently. Since honest parties generated the latter randomly and unknown to the adversary, the former must also be random and unknown to the adversary. The cost of each party sharing their $[\boldsymbol{u}_i]_{n-k}$ is $O(n)$ for a total cost of $O(n^2)$. Since $n-t = \Omega(n)$, the amortized cost is $O(n)$ per packed sharing.

The parties can repeat the same process as above, sharing random $([\boldsymbol{v}_1]_{n-k}, \ldots, [\boldsymbol{v}_n]_{n-t})$ and multiplying them by $\boldsymbol{M}$ to obtain random $([\boldsymbol{b}_1]_{n-k}, \ldots, [\boldsymbol{b}_{n-t}]_{n-k})$

with the same cost. Now, observe that for any $l \in [n-t]$,

$$\begin{aligned} \boldsymbol{c}_l \coloneqq \boldsymbol{a}_l * \boldsymbol{b}_l &= \left(\sum_{i=1}^{n} \boldsymbol{M}_l[i] \cdot \boldsymbol{u}_i\right) * \left(\sum_{j=1}^{n} \boldsymbol{M}_l[j] \cdot \boldsymbol{v}_j\right) \\ &= \sum_{i=1}^{n} \boldsymbol{M}_l[i] \cdot \left(\sum_{j=1}^{n} \boldsymbol{M}_l[j] \cdot (\boldsymbol{u}_i * \boldsymbol{v}_j)\right). \end{aligned} \tag{1}$$

Thus, a good start to getting the sharing $[\boldsymbol{c}_l]_{n-k}$, could be to securely compute $\boldsymbol{u}_i * \boldsymbol{v}_j$, for every pair of ordered parties $(P_i, P_j)$. Thankfully, this is exactly what the (programmable) OLE functionality $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ gives us [RS22]. $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ takes as input $\boldsymbol{u}_i$ from $P_i$ and $\boldsymbol{v}_j$ from $P_j$. It next samples random $\boldsymbol{\alpha}_i^j$, computes $\boldsymbol{\beta}_j^i \leftarrow \boldsymbol{u}_i * \boldsymbol{v}_j - \boldsymbol{\alpha}_i^j$, then gives $\boldsymbol{\alpha}_i^j$ to $P_i$ and $\boldsymbol{\beta}_j^i$ to $P_j$.

However, now we are faced with a dilemma. We could have every $P_i$ distribute packed Shamir secret sharings of $[\boldsymbol{\alpha}_i^j]_{n-k}$ and $[\boldsymbol{\beta}_i^j]_{n-k}$, for each other party $P_j$. Unfortunately, this would cost $\Omega(n^3)$ communication in total, which is too much. We could also have each party $P_i$ compute

$$\boldsymbol{c}_l^i \leftarrow (\boldsymbol{M}_l[i])^2 \cdot \boldsymbol{u}_i * \boldsymbol{v}_j + \boldsymbol{M}_l[i] \cdot \sum_{j=1}^{n} \boldsymbol{M}_l[j] \cdot (\boldsymbol{\alpha}_i^j + \boldsymbol{\beta}_i^j)$$

and then share $[\boldsymbol{c}_l^i]_{n-k}$. However, for every $l \in [n-t]$, this value $\boldsymbol{c}_l^i$ is different. Thus, $P_i$ needs to distribute such a sharing for each $l \in [n-t]$, which costs $\Omega(n^3)$ total that is again too much.

Indeed, although super-invertible matrices are good extractors for purely random sharings, computing packed beaver triples in this way is too expensive. Intuitively, this is because known constructions of super-invertible matrices, such as Vandermonde matrices, are *dense*. For *every* row of $\boldsymbol{M}$, *every* entry could be non-zero. This means that *every* $\boldsymbol{c}_l$ is computed from *every* party $P_i$'s $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$. Moreover, each row $\boldsymbol{M}_l$ is different (which is necessary for super-invertibility) meaning that each $\boldsymbol{c}_l$ is computed in a different way from the $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ vectors.

Thus, if we want to compute packed beaver triples in the above manner, we need *sparse* matrices $\boldsymbol{M}$. This leads us to our first key technical contribution.

**Weakly Super-Invertible Matrices.** In the above, the super-invertible matrix $\boldsymbol{M}$ enabled the property that $\boldsymbol{M}^H$ has full rank. So, the distribution of each element in $([\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_{n-k})$ is random and hidden from the adversary. This strong property is quite useful in the MPC construction, but comes at the cost of large communication for our proposed packed beaver triple construction above. The core reason is due to the density (large number of non-zero-entries) of super-invertible matrices.

Instead, we take a different approach. Suppose that $\boldsymbol{M}^H$ has high (but not necessarily full) rank. For concreteness, say the rank was $\gamma \cdot (n-t)$ as opposed to $n-t$ for some constant $0 < \gamma < 1$. Then, it will be guaranteed that a $\gamma$-fraction of $([\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_{n-k})$ will be random from the adversary's view. However, the bad $\gamma$-fraction is unknown since the corrupted parties are unknown (and could change later with an adaptive adversary). While weaker, we will later show that this still suffices for our MPC applications with an additional level of extraction when $\gamma > 1/2$.

With this weaker requirement, we construct $m \times n$ sparse matrices with only a small number of non-zero entries. We present a simple construction where for each column, $\zeta = \Theta(\log |\mathbb{F}|)$ rows are chosen uniformly at random. Then, these $\zeta$ entries in the column are set to be a uniformly random non-zero entry. The remaining $m - \zeta$ entries are set to 0. The total number of non-zero entries is $O(n \log |\mathbb{F}|)$. In our MPC application, we require larger fields $|\mathbb{F}| = O(n + 2^\lambda)$. Then, each row has $O(\lambda + \log n)$ non-zero entries except with $2^{-\lambda}$ probability.

To prove weakly super-invertibility, we analyze the probability that subsets of columns have low rank. To do this, we compute the number of vectors in the null space of the corresponding sub-matrix. We show that the null space will not contain any vectors with more than $O(\lambda + \log m)$ non-zero entries except with negligible probability implying that the sub-matrix has large rank.

**Two-level Extraction using Weakly Super-Invertible Matrices and [CP17].** Equipped with *weakly* super-invertible matrices, we can revisit our attempt to generate packed beaver triples from above. Indeed, we can still try to extract $[\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{n-t}]_m$ and $[\boldsymbol{b}_1]_{n-k}, \ldots, [\boldsymbol{b}_m]_{n-k}$ from $[\boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{u}_n]_{n-k}$ and $[\boldsymbol{v}_1]_{n-k}, \ldots, [\boldsymbol{v}_n]_{n-k}$, respectively, using *weakly* super-invertible $\boldsymbol{M}$, where $m = \Theta(n)$ (this does not fully work, as mentioned above). Now, for each $l \in [m]$, Equation 1 becomes

$$\begin{aligned}
\boldsymbol{c}_l \coloneqq \boldsymbol{a}_l * \boldsymbol{b}_l &= \left( \sum_{i \in [n] : \boldsymbol{M}_l[i] \neq 0} \boldsymbol{M}_l[i] \cdot \boldsymbol{u}_i \right) * \left( \sum_{j \in [n] : \boldsymbol{M}_l[j] \neq 0} \boldsymbol{M}_l[j] \cdot \boldsymbol{v}_j \right) \\
&= \sum_{i \in [n] : \boldsymbol{M}_l[i] \neq 0} \boldsymbol{M}_l[i] \cdot \left( \sum_{j \in [n] \setminus \{i\} : \boldsymbol{M}_l[j] \neq 0} \boldsymbol{M}_l[j] \cdot (\boldsymbol{u}_i * \boldsymbol{v}_j + \boldsymbol{u}_j * \boldsymbol{v}_i) \right) \\
&\quad + \sum_{i \in [n] : \boldsymbol{M}_l[i] \neq 0} (\boldsymbol{M}_l[i])^2 \cdot \boldsymbol{u}_i * \boldsymbol{v}_i.
\end{aligned}$$

Therefore, for each $l \in [m]$ we only need to invoke $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ between every ordered pair of parties $(P_i, P_j)$ such that $\boldsymbol{M}_l[i] \neq 0$ and $\boldsymbol{M}_j[i] \neq 0$ to receive $\boldsymbol{\alpha}_i^j$ and $\boldsymbol{\beta}_j^i$, respectively, such that $\boldsymbol{\alpha}_i^j + \boldsymbol{\beta}_j^i = \boldsymbol{u}_i * \boldsymbol{v}_j$. Based on this, each $P_i$ such that $\boldsymbol{M}_l[i] \neq 0$ can compute

$$\boldsymbol{c}_l^i \leftarrow (\boldsymbol{M}_l[i])^2 \cdot \boldsymbol{u}_i * \boldsymbol{v}_i + \boldsymbol{M}_l[i] \cdot \sum_{j \in [n] \setminus \{i\} : \boldsymbol{M}_l[j] \neq 0} \boldsymbol{M}_l[j] \cdot (\boldsymbol{\alpha}_i^j + \boldsymbol{\beta}_i^j)$$

and then share $[\boldsymbol{c}_l^i]_{n-k}$. So, all parties compute $[\boldsymbol{c}_l]_{n-k} \leftarrow \sum_{i \in [n] : \boldsymbol{M}_l[i] \neq 0} [\boldsymbol{c}_l^i]_{n-k}$.

Since the row norm of $\boldsymbol{M}$ is $O(\lambda)$ with all-but-negligible probability, we get the following two efficiency benefits. First, we only need $O(\lambda^2 \cdot m)$ invocations of $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$. Therefore, even if the $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ instantiation costs $\texttt{poly}(\kappa)$ communication per correlation, and thus $\widetilde{O}(k)$ communication per invocation, this would be a total cost of $\widetilde{O}(m \cdot n)$. Second, for each row $l \in [m]$, we only need $O(\lambda)$ parties $P_i$ to share $[\boldsymbol{c}_l^i]_{n-k}$. Thus, this only costs $O(\lambda \cdot m \cdot n)$ total communication. Therefore, in total this is $\widetilde{O}(m \cdot n)$ for $O(m)$ packed triples, or $\widetilde{O}(1)$ per underlying triple, which was our goal!

However, as noted above, since $\boldsymbol{M}$ is only *weakly* super-invertible, the extracted values $\boldsymbol{a}_l, \boldsymbol{b}_l$ may not be fully random. Indeed, we are only guaranteed that some $\gamma \cdot m$ of them are fully random, but we do not know which ones. Furthermore, we, of course, do not know the identities of the honest parties. Thus, the protocol up until this point is only our *first level* of extraction.

To obtain fully random packed beaver triples, we need a *second level* of extraction. We base this on the "triple extraction" protocol of [CP17]. This will allow us to extract $\mu \coloneqq \gamma \cdot m - (m+1)/2$ random triples via the $m$ triples obtained from the first level of extraction, some unknown $\gamma \cdot m$ of which were random. Since $\gamma > 1/2$, we have $\mu = \Omega(m) = \Omega(n)$.

Let $N = (m-1)/2$. In the second level of extraction, for each $l \in [N+1]$, the parties implicitly view the $\tau$-th secrets $a_l^\tau, b_l^\tau, c_l^\tau$ of $\boldsymbol{a}_l, \boldsymbol{b}_l, \boldsymbol{c}_l$ as the $l$-th evaluation points of polynomials $A_\tau(\cdot)$ of degree $N$, $B_\tau(\cdot)$ of degree $N$, and $C_\tau(\cdot)$ of degree $m-1$ $(= 2N)$, respectively; i.e., $A_\tau(l) = a_l^\tau$, $B_\tau(l) = b_l^\tau$, and $C_\tau(l) = c_l^\tau$. In other words, letting $\boldsymbol{A}(\cdot) \leftarrow (A_1(\cdot), \ldots, A_k(\cdot))$, $\boldsymbol{B}(\cdot) \leftarrow (B_1(\cdot), \ldots, B_k(\cdot))$, and $\boldsymbol{C}(\cdot) \leftarrow (C_1(\cdot), \ldots, C_k(\cdot))$ be vectors of polynomials, the parties have sharings

$$([\boldsymbol{A}(1)]_{n-k}, \ldots, [\boldsymbol{A}(N+1)]_{n-k}) = ([\boldsymbol{a}_1]_{n-k}, \ldots, [\boldsymbol{a}_{N+1}]_{n-k});$$

$$([\boldsymbol{B}(1)]_{n-k}, \ldots, [\boldsymbol{B}(N+1)]_{n-k}) = ([\boldsymbol{b}_1]_{n-k}, \ldots, [\boldsymbol{b}_{N+1}]_{n-k});$$

$$([\boldsymbol{C}(1)]_{n-k}, \ldots, [\boldsymbol{C}(N+1)]_{n-k}) = ([\boldsymbol{c}_1]_{n-k}, \ldots, [\boldsymbol{c}_{N+1}]_{n-k}).$$

The goal will be for the parties to compute sharings $([\boldsymbol{A}(N+2)]_{n-k}, \ldots, [\boldsymbol{A}(m)]_{n-k})$, $([\boldsymbol{B}(N+2)]_{n-k}, \ldots, [\boldsymbol{B}(m)]_{n-k})$, and $([\boldsymbol{C}(N+2)]_{n-1}, \ldots, [\boldsymbol{C}(m)]_{n-1})$, such that together with the above sharings, the underlying secret polynomial vectors $\boldsymbol{A}(\cdot)$, $\boldsymbol{B}(\cdot)$, $\boldsymbol{C}(\cdot)$ satisfy $\boldsymbol{A}(\cdot) * \boldsymbol{B}(\cdot) = \boldsymbol{C}(\cdot)$. Indeed, the first $N$ evaluation points satisfy this relation, since they come from packed beaver triples.

Now, since $\boldsymbol{A}(\cdot)$ corresponds to degree-$N$ polynomials, the $N+1$ points $[\boldsymbol{A}(1)]_{n-k}, \ldots, [\boldsymbol{A}(N+1)]_{n-k}$ define them. The same thing can be said about $\boldsymbol{B}(\cdot)$ and $[\boldsymbol{B}(1)]_{n-k}, \ldots, [\boldsymbol{B}(N+1)]_{n-k}$. Thus, we can locally compute $([\boldsymbol{A}(N+2)]_{n-k}, \ldots, [\boldsymbol{A}(m)]_{n-k})$ and $([\boldsymbol{B}(N+2)]_{n-k}, \ldots, [\boldsymbol{B}(m)]_{n-k})$ using Lagrange interpolation on the shares. To compute $([\boldsymbol{C}(l)]_{n-1}$ for $l \in [N+2, m])$, we need to compute $[\boldsymbol{A}(l) * \boldsymbol{B}(l)]_{n-1}$. We do this using the typical packed beaver multiplication procedure on input our $l$-th packed beaver triple $[\boldsymbol{a}_l]_{n-k}, [\boldsymbol{b}_l]_{n-k}, [\boldsymbol{c}_l]_{n-k}$ from the first level of extraction.[5] Since $\boldsymbol{C}(\cdot)$ corresponds to a vector of degree-$(m-1)$ polynomials, the $m$ computed points $[\boldsymbol{C}(1)]_{n-1}, \ldots, [\boldsymbol{C}(m)]_{n-1}$ define them.

Next, observe that each polynomial in the vectors $\boldsymbol{A}(\cdot)$ and $\boldsymbol{B}(\cdot)$ are of degree $N$, and thus have $N+1$ degrees of freedom, while the $\boldsymbol{C}(\cdot)$ polynomials are of degree $m-1$ and, thus, they have $m \geq N+1$ degrees of freedom. Since only

[5] This standard technique leads to $[\boldsymbol{C}(l)]_{n-1}$ having degree-$(n-1)$.

$(1-\gamma)\cdot m$ of the packed beaver triples from the first level of extraction are *not* fully random to the adversary, only that many points of the polynomials are *not* fully random to the adversary—for all other points, either nothing is directly revealed about them (if they are one of the first $N+1$ points) or they are masked by the $[\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}$ parts of a fully random beaver triple (if they are one of the last $N$ points). This means that $N+1-(1-\gamma)\cdot m = \gamma\cdot m-(m+1)/2 = \mu$ of the degrees of freedom of the polynomials in $\boldsymbol{A}(\cdot)$ and $\boldsymbol{B}(\cdot)$ are indeed fully random to the adversary, and at least as many degrees of freedom of the polynomials in $\boldsymbol{C}(\cdot)$ are also fully random to the adversary. So, we can compute using Lagrange interpolation packed Shamir sharings $([\boldsymbol{A}(m+l)]_{n-k}, [\boldsymbol{B}(m+l)]_{n-k}, [\boldsymbol{C}(m+l)]_{n-1})$ for $l \in [m+1, m+\mu]$ whose secrets correspond to $\mu$ polynomial evaluations on which the adversary has no information and thus are fully random. We output these sharings as our packed beaver triples.

The only communication required by the second level of extraction is from the $N+1 = (m+1)/2$ invocations of the packed beaver multiplication procedure. Since each of these costs $O(n)$, the whole second level costs $O(n^2)$. Therefore, our full packed beaver triple protocol costs $\widetilde{O}(n^2)$ communication for $\mu = \Omega(n)$ packed beaver triples. This is $\widetilde{O}(1)$ communication per underlying triple, and thus we have achieved our goal.

## 2.2 Offline phase of [GPS22] with $\widetilde{O}(1)$ communication per multiplication

As mentioned earlier, the main bottleneck in instantiating the offline phase of [GPS22] is generating packed beaver triples. Another challenge is efficiently generating *authenticated* packed sharings required for malicious security in the online phase. Once we have this, the rest of the offline phase can be instantiated using standard techniques (presented in Section 6 for completeness). Next, we will show that authenticating packed sharings can in fact be done using packed beaver triples.

**Authenticated Sharings with $\widetilde{O}(1)$ Communication per Value.** In the online phase of [GPS22], all circuit values will be shared using degree-$(n-k)$ *authenticated* packed Shamir sharings $[\![\boldsymbol{v}]\!]_{n-k} := ([\boldsymbol{v}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{v}]_{n-k})$, for some $\boldsymbol{\gamma} = (\gamma, \gamma \ldots, \gamma)$, where $\gamma$ is uniformly random. This $\gamma$ is referred to as the "MAC key", while the sharing $[\boldsymbol{\gamma} * \boldsymbol{v}]_{n-k}$ is referred to as the (information-theoretic) "MAC" on $\boldsymbol{v}$. Intuitively, if $\boldsymbol{v}$ is opened then it will be checked by opening $\boldsymbol{\gamma} * \boldsymbol{v}$—since $\gamma$ is random, with all-but-negligible probability, the adversary will be unable to force an opening to some $\boldsymbol{v'} \neq \boldsymbol{v}$.[6]

In the offline phase, we have to generate such authenticated packed Shamir sharings. First, we show that generating authenticated packed Shamir sharings of *random* vectors can be done using packed beaver triples. We again assume a semi-honest adversary in the offline phase, for now.

Assume that we have the sharing $[\boldsymbol{\gamma}]_{n-k}$ of some random $\boldsymbol{\gamma}$ (this can be generated using standard techniques) and some packed beaver triple $([\boldsymbol{r}]_{n-k}, [\boldsymbol{s}]_{n-k}, [\boldsymbol{r}*$

[6] This is done once for many such openings at a time using random linear combinations.

$\boldsymbol{s}]_{n-k}$). First, the parties can computes $[\boldsymbol{\gamma} + \boldsymbol{s}]_{n-k} \leftarrow [\gamma]_{n-k} + [\boldsymbol{s}]_{n-k}$ and open their shares to $P_1$. Since $\boldsymbol{s}$ is random, nothing about $\boldsymbol{\gamma}$ is revealed. $P_1$ can then reconstruct $\boldsymbol{\gamma} + \boldsymbol{s}$, compute the (unique) packed Shamir sharing $[\boldsymbol{\gamma} + \boldsymbol{s}]_{k-1}$, and distribute to the rest of the parties their shares. Finally, all parties can compute $[\boldsymbol{\gamma} * \boldsymbol{r}]_{n-1} \leftarrow [\boldsymbol{r}]_{n-k} * [\boldsymbol{\gamma} + \boldsymbol{s}]_{k-1} - [\boldsymbol{r} * \boldsymbol{s}]_{n-k}$. Observe that the parties opening their shares of $[\boldsymbol{\gamma} + \boldsymbol{s}]_{n-k}$ to $P_1$, and then $P_1$ distributing the shares of $[\boldsymbol{\gamma} + \boldsymbol{s}]_{k-1}$ each cost $O(n)$ communication. So, producing $[\boldsymbol{\gamma} * \boldsymbol{r}]_{n-1}$ costs additional $O(1)$ communication per underlying value. Using standard techniques, the parties can obtain a sharing $[\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k}$ with reduced degree for the same cost (see Section 6).

Once we have authenticated packed Shamir sharings $[\![\boldsymbol{r}]\!]_{n-k}$ of random vectors $\boldsymbol{r}$, we can produce authenticated packed Shamir sharings $[\![\boldsymbol{x}]\!]_{n-k}$ of any vector $\boldsymbol{x}$ (including components of packed beaver triples) efficiently using standard techniques. We refer the reader to Section 6 for more details.

**Malicious Security.** For malicious security of the offline phase, we can use several standard techniques that we will instantiate in Section 6. One primary opportunity for the corrupted parties to deviate from the protocol is in generating the packed beaver triples. For example, a corrupt party $P_j$ can input to $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ different vectors $\overline{\boldsymbol{u}}_j, \overline{\boldsymbol{v}}_j$ than those underlying their sharings $[\boldsymbol{u}]_{n-k}, [\boldsymbol{v}]_{n-k}$. The adversary can use this and other methods so that the output authenticated packed beaver triples will actually be of the form $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{a}*\boldsymbol{b}+\boldsymbol{\delta}]\!]_{n-k})$, for some $\boldsymbol{\delta}$. To handle this, the parties can use the standard *triple sacrificing* technique [DPSZ12] which takes as input two packed beaver triples $([\![\boldsymbol{a}_1]\!]_{n-k}, [\![\boldsymbol{b}_1]\!]_{n-k}, [\![\boldsymbol{a}_1*\boldsymbol{b}_1+\boldsymbol{\delta}_1]\!]_{n-k})$ and $([\![\boldsymbol{a}_2]\!]_{n-k}, [\![\boldsymbol{b}_2]\!]_{n-k}, [\![\boldsymbol{a}_2 * \boldsymbol{b}_2 + \boldsymbol{\delta}_2]\!]_{n-k})$, then outputs the first if $\boldsymbol{\delta}_1 = \boldsymbol{\delta}_2 = \mathbf{0}$. This standard procedure uses only $O(1)$ overhead per underlying triple.

**Additional Preprocessing Required for [EGP$^+$23].** Although a major bottleneck in instantiating the offline phase of [EGP$^+$23] is also packed beaver triples, their offline phase has another major roadblock that is not present in that of [GPS22]. The online phase of [EGP$^+$23] associates some value $\lambda_\gamma \in \mathbb{F}$ with every circuit wire $\gamma$. In particular, if $\gamma$ is the output wire of a multiplication gate, then the value $\lambda_\gamma$ associated with it is chosen randomly. Additionally, for every group of $k$ multiplication gates in the circuit with input wires $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the online phase requires packed secret sharings $[\boldsymbol{\lambda}_\alpha]_{n-k}$ and $[\boldsymbol{\lambda}_\beta]_{n-k}$ which share the values corresponding to those wires. Thus, if a given multiplication output wire $\gamma$ is contained in some later vector of multiplication input wires $\boldsymbol{\alpha}$, then $[\boldsymbol{\lambda}_\alpha]_{n-k}$ should secret share $\lambda_\gamma$ in the corresponding slot. However, if the multiplication gate for which $\gamma$ is the output wire has fan-out $f > 1$, then $\lambda_\gamma$ may have to be secret shared in several different $[\boldsymbol{\lambda}_{\alpha_1}]_{n-k}, [\boldsymbol{\lambda}_{\alpha_2}]_{n-k}, \ldots, [\boldsymbol{\lambda}_{\alpha_f}]_{n-k}$ such that $\boldsymbol{\lambda}_{\alpha_1} \neq \boldsymbol{\lambda}_{\alpha_2} \neq \cdots \neq \boldsymbol{\lambda}_{\alpha_f}$.

Unfortunately, using the aforementioned technique of sampling several random *independent* packed Shamir sharings at a time with $O(n)$ amortized communication per sharing is insufficient for this. It is unclear how to adapt this technique to get several *correlated* packed Shamir sharings $[\boldsymbol{\lambda}_{\alpha_1}]_{n-k}, [\boldsymbol{\lambda}_{\alpha_2}]_{n-k}, \ldots, [\boldsymbol{\lambda}_{\alpha_f}]_{n-k}$ with amortization. Indeed, the offline protocol of [EGP$^+$23] requires $\Omega(n^2)$ communication per sharing to achieve this.

## 3 Preliminaries

We now present some important preliminaries. For additional preliminaries on additive secret sharing and basic ideal functionalities $\mathcal{F}_{\mathsf{commit}}$ and $\mathcal{F}_{\mathsf{coin}}$ that we will use in our MPC protocol, see Appendix A of the Supplementary Material.

**Linear Algebra.** We denote a $k$-dimensional vector as $\boldsymbol{v} = (v_1, \dots, v_k)^\intercal \in \mathbb{F}^k$. Given two $k$-dimensional vectors $\boldsymbol{u}, \boldsymbol{v}$, we use $\boldsymbol{u} * \boldsymbol{v}$ to denote component-wise multiplication of the vectors.

For any $m \times n$ matrix $\boldsymbol{M}$, we denote its $i$-th row vector as $\boldsymbol{M}_i$ for any $i \in [m]$. The entry in the $i$-th row and $j$-th column of $\boldsymbol{M}$ is denoted by $\boldsymbol{M}_i[j]$ for any $i \in [m]$ and $j \in [n]$. We will denote the $j$-th column vector of $\boldsymbol{M}$ as $\boldsymbol{M}^j$ for any $j \in [n]$. For convenience, we will overload these operators to consider subsets of rows or columns of a matrix. For example, let $C = \{c_1, \dots, c_\ell\} \subseteq [n]$ be any subset of columns and we define the following $m \times |S|$ sub-matrix $\boldsymbol{M}^C = [\boldsymbol{M}^{c_1}, \dots, \boldsymbol{M}^{c_\ell}]$ consisting of all column vectors of $\boldsymbol{M}$ in the subset of columns $C$. Finally, we can consider sub-matrices of $\boldsymbol{M}$ that are defined by arbitrary subsets of both rows and columns. For any subset of rows $R = \{r_1, \dots, r_k\} \subseteq [m]$ and subset of columns $C = \{c_1, \dots, c_\ell\} \subseteq [n]$, we define the following $k \times \ell$ sub-matrix

$$\boldsymbol{M}_R^C = \begin{bmatrix} \boldsymbol{M}_{r_1}[c_1] & \dots & \boldsymbol{M}_{r_1}[c_\ell] \\ \dots & \dots & \dots \\ \boldsymbol{M}_{r_k}[c_1] & \dots & \boldsymbol{M}_{r_m}[c_\ell] \end{bmatrix}$$

consisting all entries that appear in one of the rows denoted by $R$ and one of the columns denoted by $C$. We will use $\boldsymbol{M}^\intercal$ to denote the transpose matrix of $\boldsymbol{M}$.

**MPC Preliminaries.** In this paper, we focus on using MPC with $n$ parties $P_1, \dots, P_n$ to compute functionalities $\mathcal{F}$ that can be represented as arithmetic circuits over a finite field $\mathbb{F}$ with input, addition, multiplication, and output gates. We use $\kappa$ to denote the computational security parameter, $\lambda$ to denote the statistical security parameter, $C$ to denote the circuit, and $|C|$ for the number of multiplication gates in the circuit. In this work, we assume that the field size is $|\mathbb{F}| \geq 2^\lambda$. Note that this implies that $|F| \geq |C| + n$, since both the number of parties and circuit size are bounded by $\mathtt{poly}(\lambda)$.

We consider an adversary $\mathcal{A}$ that can corrupt at most $t$ parties adaptively and maliciously. We study the $t < (1 - \varepsilon) \cdot n$ setting, where $\varepsilon$ is any constant $0 < \varepsilon \leq 1/2$. Such an adversary receives all messages sent to currently corrupted parties and based on these can *adaptively* corrupt more parties, as long as the number of corrupted parties does not exceed $t$. In addition, the adversary can *maliciously* arbitrarily deviate from the protocol.

Ultimately, we will study MPC protocols in the offline-online setting in which there is an offline phase that is *circuit-independent* and is *computationally*-secure with respect to $\kappa$, followed by an online setting that can be *statistically*-secure with respect to $\lambda$. In this paper, we instantiate the offline phase of an MPC protocol. However, we will assume the presence of an ideal functionality which can prepare circuit-independent correlated randomness (OLE correlations). Therefore, we can still prove our protocols *statistically*-secure.

We use the Universal Composability (UC) framework of [Can01] to formally prove security of our protocol. In this framework, we have the following two worlds:

– **Real World Execution.** In the real world, the adversary $\mathcal{A}$ interacts with honest parties and can adaptively corrupt up to $t$ of them. At the end of the protocol, the output of the real-world execution includes the inputs and outputs of honest parties and the view of the adversary.
– **Ideal World Execution.** In the ideal world, a simulator $\mathcal{S}$, simulates honest parties and interacts with the adversary $\mathcal{A}$. Furthermore, $\mathcal{S}$ has one-time access to $\mathcal{F}$, which includes providing inputs of corrupted parties to $\mathcal{F}$, receiving the outputs of corrupted parties, and sending instructions specified in $\mathcal{F}$ (e.g., asking $\mathcal{F}$ to abort). The output of the ideal world execution includes the inputs and outputs of honest parties and the view of the adversary.

In more detail, we say that a protocol $\Pi$ UC-realizes $\mathcal{F}$ if there exists a simulator $\mathcal{S}$, such that for every adversary $\mathcal{A}$, the distribution of the output of the real world execution is (in our case, statistically) indistinguishable from the distribution of the output of the ideal world execution.

**Packed Shamir Secret Sharings.** In our work, we will use *packed Shamir secret sharing*, introduced by [FY92]. This is a generalization of standard Shamir secret sharing [Sha79]. With packed Shamir secret sharing, we will pack $k$ secrets into one sharing. A *degree-$d$* ($d \geq k-1$) packed Shamir sharing of $\boldsymbol{x} = (x_1, \ldots, x_k)^\intercal \in \mathbb{F}^k$ is a vector $(w_1, \ldots, w_n)$ for which there is some polynomial $f(x)$ of degree at most $d$ such that $f(-i+1) = x_i$ for all $i \in [k]$ and $f(i) = w_i$ for all $i \in [n]$. We denote such a sharing $[\boldsymbol{x}]_d$. The $i$-th share $w_i$ is held by $P_i$. Reconstructing a degree-$d$ packed Shamir sharing requires $d+1$ shares and can be done using Lagrange interpolation. For a random degree-$d$ packed Shamir sharing of $\boldsymbol{x}$, any $d-k+1$ shares are independent of the secret $\boldsymbol{x}$. We therefore say that a degree-$d$ packed Shamir sharing is *private* against an adversary that holds $d-k+1$ shares. For some $C \subseteq [n]$, we say that some sharing $[\boldsymbol{x}]_d$ is *consistent* with set $\{v_i\}_{i \in C}$ if $w_i = v_i$ for all $i \in C$.

Packed Shamir sharings have the following properties, which follow directly from the computation of the underlying polynomials:

– Linearity: For any $n > d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x}+\boldsymbol{y}]_d = [\boldsymbol{x}]_d + [\boldsymbol{y}]_d$.
– Multiplicative: For any $d_1, d_2 \geq k-1$ such that $d_1 + d_2 < n$ and any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x} * \boldsymbol{y}]_{d_1+d_2} = [\boldsymbol{x}]_{d_1} * [\boldsymbol{y}]_{d_2}$, where $*$ is component-wise multiplication.

Note that the second property implies that for all $k-1 \leq d \leq n-k$, a degree-$d$ packed Shamir secret sharing is *multiplication-friendly.* What we mean by this is that for all $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{F}^k$, all parties can locally compute $[\boldsymbol{c} * \boldsymbol{x}]_{d+k-1}$ from $[\boldsymbol{x}]_d$ and public vector $\boldsymbol{c}$. To do this, all parties just locally transform $\boldsymbol{c}$ to the (unique) degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{c}]_{k-1}$ and then use the $*$ operation.

Recall that $t$ is the maximal number of corrupted parties and that a degree-$d$ packed Shamir secret sharing is private against an adversary that holds $d-k+1$

shares. To ensure that the Shamir secret sharing is both private and multiplication-friendly, we choose $k$ such that $t \leq d - k + 1$ and $d \leq n - k$. When $d = n - k$ and $k = (n - t + 1)/2$, both requirements hold and $k$ is maximal.

## 4 Revisiting Invertible Matrices

Many prior MPC works have utilized matrices that emit certain strong invertibility properties. These matrices have been found in applications as sub-routines of MPC protocols such as critically being able to extract many random sharings from some base random sharings (some of which are adversarially generated).

**Super-Invertible Matrices.** First defined by Hirt and Nielsen [HN06], super-invertible matrices are any matrix $\boldsymbol{M}$ of dimension $m \times n$ where the number of columns is at least as large as the number of rows (that is, $n \geq m$). The matrix $\boldsymbol{M}$ is super-invertible if every sub-matrix of dimension $m \times m$ is invertible. In other words, for any subset of $m$ columns, the sub-matrix consisting of the $m$ chosen columns should have full rank. We present the formal definition below:

**Definition 1 (Super-Invertible Matrices [HN06]).** *Let $\boldsymbol{M}$ be a matrix of $m \times n$ dimension with $n \geq m$. $\boldsymbol{M}$ is a super-invertible matrix if, for any subset $C = \{c_1, \ldots, c_m\} \subseteq [n]$ of size exactly $m$, the $m \times m$ sub-matrix $\boldsymbol{M}^C$, defined as consisting of the $m$ column vectors of $\boldsymbol{M}$ in the subset $C$, must be invertible.*

**Hyper-Invertibile Matrices.** Hyper-invertible matrices were introduced by Beerliová-Trubíniová and Hirt [BTH08]. In super-invertible matrices, it was only required that every $m \times m$ sub-matrix was full rank $m$. For hyper-invertible matrices, the invertibility requirement is extended to apply for every square sub-matrix. In particular, for a $m \times n$ matrix $\boldsymbol{M}$ to be hyper-invertible, it must be that every $z \times z$ sub-matrix of $\boldsymbol{M}$ must be invertible for every choice of $1 \leq z \leq m$. In other words, for every $1 \leq z \leq m$ and every choice of subsets $R \subseteq [m]$ and $C \subseteq [n]$ both of exactly size $z$, the sub-matrix $\boldsymbol{M}_R^C$ consisting of the rows in $R$ and columns in $C$ must be invertible (have full rank $z$). See Appendix B.2 for the definition.

**Existing Constructions.** Both super-invertible and hyper-invertible matrices have been used extensively in multiple MPC protocols. In general, both of these matrices have been instantiated using Vandermonde matrices that can be shown to satisfy the requirements of being hyper-invertible (and, thus, also being super-invertible). We refer readers to prior works (such as [HN06, BTH08]) for more details on Vandermonde matrices being super- and hyper-invertible. However, we point out that Vandermonde matrices of dimension $m \times n$ will consist of $\Omega(m \cdot n)$ non-zero entries that has a direct impact on the communication efficiency of MPC protocols. To our knowledge, we are unaware of any super-invertible or hyper-invertible matrices with $o(m \cdot n)$ non-zero entries. In fact, we show hyper-invertible matrices with $o(m \cdot n)$ non-zero entries cannot exist later (see Appendix B.3).

## 5 Weakly Super-Invertible Matrices

In this section, we present new, weaker notions of super-invertible matrices with less restrictive requirements compared to super-invertible matrices. We later show that the benefit of considering weaker invertibility notions is that we obtain constructions with a significantly smaller number of non-zero entries that can improve the efficiency when used in MPC applications.

### 5.1 Definition

Our new definitions of $m \times n$ *weakly super-invertible matrices* will weaken definitions in two ways. First, we will no longer require $m \times m$ sub-matrices to be invertible (that is, equivalent to being full rank). Instead, we will only require that sub-matrices of interest will have sufficiently high rank (but not necessarily full rank). We will desire that $m \times h$ sub-matrices should have rank sufficiently close to $m$ for sufficiently large $h = \Omega(m)$. In particular, we will require that the rank of each $m \times h$ sub-matrix should have rank $\gamma \cdot m$ for some constant $\gamma > 1/2$. The requirement of $\gamma > 1/2$ will be important for our MPC applications later.

Secondly, we will consider probabilistic guarantees for matrices that are randomly generated. To do this, we will consider random matrix families $\mathcal{M}$ equipped with an algorithm to randomly generate some matrix from the family. At a high level, weakly super-invertible matrix families will require that, for any randomly generated matrix $m \times n$, $\boldsymbol{M}$, from the family $\mathcal{M}$, every $m \times h$ sub-matrix of $\boldsymbol{M}$ for sufficiently large $h = \Omega(m)$ should have rank at least $\gamma \cdot m$ for some constant $\gamma > 1/2$ except with negligible probability where the randomness is over the choice of the original generated matrix from the family $\mathcal{M}$.

**Matrix Families.** To define weakly super-invertible matrix families, we start by defining the notion of a matrix family $\mathcal{M}$ that is equipped with a randomized algorithm to generate random matrices from the family $\mathcal{M}$. All matrices in the family $\mathcal{M}$ will have the same dimension of $m \times n$ and every family $\mathcal{M}$ will be defined with respect to some fixed field $\mathbb{F}$. We define a matrix family as follows:

**Definition 2 (Matrix Family).** *Let $\mathbb{F}$ be some field. A matrix family of $m \times n$ matrices is the tuple $\mathcal{M} = (\mathcal{S}, \mathsf{Gen})$ satisfying the following:*

- *$\mathcal{S} \subseteq \mathbb{F}^{m \times n}$ is a subset of all $m \times n$ matrices.*
- *$\boldsymbol{M} \leftarrow \mathsf{Gen}(R)$ takes a random string $R$ as input and outputs a matrix $\boldsymbol{M} \in \mathcal{S}$.*

**Weakly Super-Invertible Matrix Families.** Using matrix families, we can now define our new notion of weakly super-invertible matrix families that extends the super-invertibility notion by Hirt and Nielsen [HN06].

In our definition, we consider the random process where a random matrix $\boldsymbol{M} \leftarrow \mathsf{Gen}(R)$ is generated from a matrix family $\mathcal{M}$ for a uniformly random chosen $R$. Afterwards, pick any subset of $m$ columns $C \subseteq [n]$ and consider the sub-matrix $\boldsymbol{M}^C$ consisting of the $m$ column vectors of $\boldsymbol{M}$ denoted by $C$. We say that a matrix family $\mathcal{M}$ is weakly super-invertible if every choice of $h = \Omega(m)$ columns $C$ results in a sub-matrix $\boldsymbol{M}^C$ that has rank at least $\Omega(m)$ except with negligible probability over the random choice of the matrix $\boldsymbol{M}$.

**Definition 3 (Weakly Super-Invertible Matrix Family).** *Fix constants $0 < \delta, \gamma, \eta < 1$ such that $\eta \geq \gamma$. Let $\mathcal{M} = (\mathcal{S}, \mathsf{Gen})$ be a matrix family over $m \times n$ matrices where $n \geq m$. $\mathcal{M}$ is $(\delta, \gamma, \eta)$-weakly super-invertible if*

$$\Pr_{\mathbf{R}}[\exists C \subseteq [n] \mid \mathsf{rank}(\boldsymbol{M}^C) < \gamma \cdot m, \boldsymbol{M} \leftarrow \mathsf{Gen}(\mathbf{R}), |C| = \eta \cdot m] \leq \delta.$$

In other words, a weakly super-invertible matrix guarantees that no $m \times (\eta \cdot m)$ sub-matrix will have rank strictly less than $\gamma \cdot m$ except with probablity $\delta$. We note that the restriction of $\eta \geq \gamma$ is necessary as we want the sub-matrix to have rank at least $\gamma \cdot m$. Therefore, the number of columns in the sub-matrix must satisfy $\eta \cdot m \geq \gamma \cdot m$ implying $\eta \geq \gamma$.

Finally, we define the main notions of efficiency for matrix families that will be important in our MPC applications. In particular, we will desire that all the matrices in some matrix family $\mathcal{M}$ have a small number of non-zero entries. In particular, we want the guarantee that each row of $\mathcal{M}$ has a small number of non-zero entries (typically, much less than $n$). For the $i$-th row vector of $\boldsymbol{M}$, we define its $L_0$ norm as $L_0(\boldsymbol{M}_i) = |\{j \mid j \in [n], \boldsymbol{M}_i[j] \neq 0\}|$. We formally define this efficiency measure for weakly super-invertible matrix families as follows:

**Definition 4 ($(\delta, \gamma, \eta, \ell)$-Weakly Super-Invertible Matrix Family).** *Let $\mathcal{M} = (\mathcal{S}, \mathsf{Gen})$ be a matrix family over $m \times n$ matrices. The matrix family $\mathcal{M}$ is $(\delta, \gamma, \eta, \ell)$-weakly super-invertible if the following holds:*

1. *$\mathcal{M}$ is $(\delta, \gamma, \eta)$-weakly super-invertible.*
2. *The probability that there exists any row vector with $L_0$ norm larger than $\ell$ is at most $\delta$. In other words, $\Pr[\exists i \in [m] \mid L_0(\boldsymbol{M}_i) > \ell] \leq \delta$.*

## 5.2 Sparse Weakly Super-Invertible Matrices

We present sparse weakly super-invertible $m \times n$ matrices over any finite field $\mathbb{F}$ with $O(n \log |\mathbb{F}|)$ non-zero entries. For constant-sized fields $|\mathbb{F}| = O(1)$, $\mathcal{M}_{\zeta\mathbf{col}}$ has $O(n)$ non-zero entries that we will later show is asymptotically optimal. For our MPC applications, we will consider larger fields such that the $L_0$ norm of $\mathcal{M}_{\zeta\mathbf{col}}$ is optimal except for the $O(\log |\mathbb{F}|)$ factor. To our knowledge, all previously known super-invertible matrices have $\Theta(m \cdot n)$ non-zero entries such as the Vandermonde matrix (see Section 4).

**Construction.** First, we will fix any finite field $\mathbb{F}$. We start by presenting our construction of the matrix family $\mathcal{M}_{\zeta\mathbf{col}} = (\mathcal{S}_{\zeta\mathbf{col}}, \mathsf{Gen}_{\zeta\mathbf{col}})$ whose matrix generation algorithm is defined as follows:

$\mathsf{Gen}_{\zeta\mathbf{col}}(\mathbf{R})$**:**

1. Initialize $m \times n$ matrix $\boldsymbol{M}$ to be all zeroes.
2. For each $i = 1, \ldots, n$:
   (a) Use $\mathbf{R}$ to generate $\zeta$ integers $x_1^i, x_2^i, \ldots, x_\zeta^i$ uniformly at random from $[m]$ and $\zeta$ non-zero integers $y_1^i, \ldots, y_\zeta^i$ uniformly at random from $\mathbb{F} \setminus \{0\}$. Set $\boldsymbol{M}_{x_j^i}[i] = y_j^i$ for every $j \in \{1, 2, \ldots, \zeta\}$.

3. Return $\boldsymbol{M}$.

In other words, $\mathcal{M}_{\zeta\mathbf{col}}$ generates matrices by picking a uniformly random subset of $\zeta$ entries for each column vector and setting them to be uniformly random non-zero element in $\mathbb{F}$. The subset $\mathcal{S}_{\zeta\mathbf{col}}$ consists of all matrices with at most $\zeta$ non-zero entries in each of the columns and zero entries in all the remaining entries of each column. Note, it is possible that a column picks the same row twice. Therefore, there are at most $\zeta$ non-zero entries in each column. Next, we show that our construction, $\mathcal{M}_{\zeta\mathbf{col}}$, is weakly super-invertible:

**Theorem 4.** *Pick any choice of constants $c \geq 1$, $0 < \gamma < 1$ and $\gamma < \eta \leq c$ as well as finite field $\mathbb{F}$ where $|\mathbb{F}| \geq 3$. Let the number of columns be $n = c \cdot m$ and the number of non-zero entries in each column be $\zeta = \Theta(\log |\mathbb{F}|)$. The $m \times n$ matrix family $\mathcal{M}_{\zeta\mathbf{col}}$ is $(2^{-\lambda}, \gamma, \eta, O(\lambda + \log m))$-weakly super-invertible over $\mathbb{F}$ for sufficiently large $m = \Omega(\lambda \log m)$.*

In this section, we will focus on fields where $|\mathbb{F}| \geq 3$. Note, the requirements of the field size $|\mathbb{F}|$ is less restrictive than standard MPC protocol requirements. For example, it is commonly required that $|\mathbb{F}| \geq n$ if $n$ is the number of parties to enable (packed) Shamir secret sharing. Indeed, the number of columns $n$ will correspond exactly to the number of parties in our MPC construction. Similarly, MPC security commonly requires that each element of $\mathbb{F}$ is $\lambda$ bits meaning that $|\mathbb{F}| \geq 2^{\lambda}$. While we use such large fields for our MPC protocol, we show that constant field sizes $|\mathbb{F}| = O(1)$ are sufficient for weakly super-invertibility.

In Appendix B, we show our constructions in fact hold for all finite fields by showing that $\mathcal{M}_{\zeta\mathbf{col}}$ remains weakly super-invertible when $|\mathbb{F}| = 2$. In this case, the proof differs and we rely on prior work for the XORSAT problem [PS16].

We start with efficiency proving the expected and worst case number of non-zero entries ($L_0$ norm) of each row:

**Theorem 5.** *Let $\boldsymbol{M}$ be a random $m \times n$ matrix generated by $\mathcal{M}_{\zeta\mathbf{col}}$ with $\zeta$ non-zero entries in each column. For all rows $i \in [m]$, $\mathbb{E}[L_0(\boldsymbol{M}_i)] = O(\zeta)$. Also, for some $\ell = O(\max(\zeta, \lambda + \log m))$, $\Pr[\exists i \in [m] \mid L_0(\boldsymbol{M}_i) > \ell] \leq 2^{-\lambda}$.*

*Proof.* Each of the $n$ column vectors has exactly $\zeta$ non-zero entries. So, the expected $L_0$ norm for any row is $\zeta \cdot (n/m) = O(\zeta)$ as $n = O(m)$.

For the worst case bound, fix any row $i \in [m]$. For any $j \in [n]$, let $X_j$ be the random binary variable indicating whether the $j$-th column has a non-zero entry in the $i$-th row. Then, we see that $\Pr[X_j] = \zeta/m$. So, $\mu = \mathbb{E}[X_1 + \ldots + X_n] = n\zeta/m = c\zeta$ for the chosen constant $c \geq 1$ that is the sum of $n$ independent binary variables. By Chernoff's bound, we get that the probability that the $i$-th row vector has more than $\ell = O(\max(\zeta, \lambda + \log m))$ non-zero entries is at most $2^{-\lambda - \log m}$ for sufficiently large $\ell$. Finally, we apply a Union bound over all $m$ rows to obtain the probability upper bound of $2^{-\lambda}$. □

Note, if one plugs in $\zeta = \Theta(\log |\mathbb{F}|)$, then we see that each row has expected $O(\log |\mathbb{F}|)$ non-zero entries. In the worst case, each row has at most

$\ell = O(\max(\log|\mathbb{F}|, \lambda + \log m))$ non-zero entries. As discussed earlier, it is typically assumed that $|F| \geq n = \Omega(m)$ and $|F| \geq 2^{\lambda}$. Therefore, this means that $\ell = O(\lambda + \log m)$ as both arguments in the max are the same.

The remaining of this section will be dedicated to proving that for $h = \eta \cdot m$, all $m \times h$ sub-matrices have rank at least $\gamma \cdot m$ except with probability $\delta \leq 2^{-\lambda}$.

**Proof Overview of High Rank Sub-Matrices.** We first present a high-level overview of our proof that will show that a randomly generated matrix from $\mathcal{M}_{\zeta\mathbf{col}}$ will satisfy the necessary requirements of being weakly super-invertible except with probability $2^{-\lambda}$. Our proof will use the following steps. In the following, we will use "size" of a vector to refer to its number of non-zero entries (that is, the $L_0$ norm of the vector).

1. First, we consider vectors in the null space and show that the size of the null space can be used to lower bound the rank of sub-matrices.
2. We consider the case where $m \times h$ sub-matrices may contain null space vectors of small sizes. To analyze this case, we consider a combinatorial argument showing that null space vectors of size $x$ satisfying $x = \Omega(\lambda + \log m)$ and $x = O(m/\zeta)$ do not exist except with negligibly small probability.
3. To show that larger null space vectors do not exist with all-but-negligible probability, we show that larger null space vectors will likely consider larger sets of rows that are unlikely to emit linear combinations summing to the zero vector.
4. We put everything together to prove that $\mathcal{M}_{\zeta\mathbf{col}}$ is weakly super-invertible.

**Step 1: Null Space Vectors and Rank.** To analyze our matrix construction, we will consider null space vectors. Consider any random $m \times n$ matrix $\boldsymbol{M}$ generated by $\mathcal{M}_{\zeta\mathbf{col}}$. We say that a vector $(a_1, \ldots, a_n) \in \mathbb{F}^n$ is in the null space of $\boldsymbol{M}$ if

$$a_1\boldsymbol{M}^1 + a_2\boldsymbol{M}^2 + \ldots + a_n\boldsymbol{M}^n = \mathbf{0}.$$

For convenience, we will focus on the null space vectors' non-zero entries and their corresponding non-zero coefficients. Consider any subset of columns $C = \{i_1, \ldots, i_x\}$ such that there exists some linear combination consisting of non-zero coefficients $a_1, \ldots, a_x \in \mathbb{F} \setminus \{0\}$ such that their component-wise sum is the all-zero vector in the field $\mathbb{F}$; i.e.,

$$a_1\boldsymbol{M}^{i_1} + a_2\boldsymbol{M}^{i_2} + \ldots + a_x\boldsymbol{M}^{i_x} = \mathbf{0}.$$

In this phrasing, the corresponding null space vector consists of the subset of $x$ columns $C$ and the corresponding coefficients $a_1, \ldots, a_x$. Therefore, the size ($L_0$ norm) of a null space vector is equal to the size of the column subset. The number of null space vectors can be used to directly bound the rank of the matrix $\boldsymbol{A}$ as follows:

**Lemma 1.** *Consider any matrix $\boldsymbol{A} \in \mathbb{F}^{m \times n}$ with $h \leq m$ and the number of null space vectors is $Y$. Then, the rank of $\boldsymbol{A}$ over finite field $\mathbb{F}$ is exactly $h - \log_{|\mathbb{F}|}(Y)$.*

*Proof.* Note that the null space is a subspace of $\mathbb{F}^h$. Therefore, the dimension of the null space is exactly $\log_{|\mathbb{F}|}(Y)$. By the rank-nullity theorem, we know that rank of $\boldsymbol{A}$ and dimension of the null space of $\boldsymbol{A}$ must sum to the number of columns of $\boldsymbol{A}$. Therefore, the rank of $\boldsymbol{A}$ over $\mathbb{F}$ is exactly $r = h - \log_{|\mathbb{F}|}(Y)$. □

We use $Y^x$ to denote the number of null space vectors of size $x$ with exactly $x$ non-zero entries. We will break down the analysis into considering the number of null space vectors of size $x$, $Y^x$, for different sizes: $x \leq \beta \cdot m/\zeta = O(m/\zeta)$ and any $x = \Omega(m/\zeta)$. Each of the next two steps will handle each of these cases.

**Step 2: Small Null Space Vectors.** We will consider null space vectors of size at most $x \leq \beta \cdot m/\zeta$ for some constant $\beta > 0$ we will pick later. Consider any matrix $\boldsymbol{M}$ generated by the matrix family $\mathcal{M}_{\zeta\mathbf{col}}$. We start by showing that $\boldsymbol{M}$ will not contain many null space vectors of size at most $x \leq \beta \cdot m/\zeta$. To do this, consider any fixed subset of $x$ columns, $C \subseteq [n]$. Each column picks at most $\zeta$ rows to place non-zero entries. Consider all $t$ rows that are chosen at least once across the $x$ columns. If $C$ is part of a null space vector, we show that each of the $t$ rows must be chosen at least twice. Therefore, we immediately see that $t \leq \zeta x/2$ as a total of $\zeta x$ rows are chosen across the $x$ columns. We bound the probability that each row is chosen at least twice in the following lemma:

**Lemma 2.** *Pick any constant $c \geq 1$. Let $n = c \cdot m$. Let $\boldsymbol{M}$ be a $m \times n$ matrix generated by $\mathcal{M}_{\zeta\mathbf{col}}$. Then, there exists constant $0 < \beta < 1$ and $c_x = \Omega(\lambda + \log m)$ such that the probability there exists at least one null space vector of any size $c_x \leq x \leq \beta \cdot m/\zeta$ in $\boldsymbol{M}$ is at most $2^{-2\lambda}$ for sufficiently large $m = \Omega(\lambda)$.*

*Proof.* We first fix any $c_x \leq x \leq \beta \cdot m/\zeta$ number of columns and will show that the probability that any subset of $x$ columns forms a null space vector is at most $2^{-\Omega(\zeta x)}$. From this, we can conclude that for every number $\beta \cdot m/\zeta \geq x \geq c_x$ of rows, for sufficiently large $c_x = \Omega(\lambda + \log m)$ such that $2^{-\Omega(\zeta x)} \leq 2^{-\Omega(\zeta c_x)} \leq 2^{-2\lambda - \log m}$, the probability that there exists a null space vector of any size $x$ is at most

$$\sum_{x=c_x}^{\beta \cdot m/\zeta} 2^{-\Omega(\zeta x)} \leq (\beta \cdot m/\zeta) \cdot 2^{-2\lambda - \log m} \leq 2^{-2\lambda}.$$

Note, we can choose such $c_x = \Omega(\lambda + \log m)$ since $m = \Omega(\lambda)$ is sufficiently large.

To show this bound on the above probability, fix any subset of $x$ columns. Let $t$ be the number of rows chosen by at least one of the $x$ columns. For this subset of columns to be part of a null space vector, it must be that each row is chosen at least twice. If a row is chosen exactly once, then we can immediately see that it is impossible to obtain the zero vector as the product of two non-zero elements in $\mathbb{F}$ will always be non-zero since $\mathbb{F}$ is a field. Therefore, we know that $t \leq \zeta x/2$. Fix any subset of $t$ rows chosen by the $x$ columns and we will calculate an upper bound on the probability that these $x$ columns form a null space vector over these $t$ rows. This is upper bounded by the probability that each column only chooses from these $t$ variables. Therefore, we get that this probability is at most $(t/m)^\zeta$ for each of the $x$ columns independently and the probability is $(t/m)^{\zeta x}$

for all $x$ columns. Next, we perform a Union bound over all possible choices of the $x$ columns and the $t$ rows to get

$$\binom{n}{x}\binom{m}{t}\left(\frac{t}{m}\right)^{\zeta x} \leq \left(\frac{en}{x}\right)^{x} \cdot \left(\frac{em}{t}\right)^{t} \cdot \left(\frac{t}{m}\right)^{\zeta x}$$

by applying the upper bound that $\binom{a}{b} \leq (ea/b)^b$. The above probability is strictly increasing in $t$ and since $t \leq \zeta x/2$, the probability is at most

$$\left(\frac{en}{x}\right)^{x}\left(\frac{2em}{\zeta x}\right)^{\zeta x/2}\left(\frac{\zeta x}{2m}\right)^{\zeta x} \leq \left(\frac{e^{\zeta/2+1} n x^{\zeta/2-1}\zeta^{\zeta/2}}{2^{\zeta/2} m^{\zeta/2}}\right)^{x} = O\left(\frac{x\zeta^{\zeta/(\zeta-2)}}{m}\right)^{(\zeta/2-1)\cdot x}.$$

Note that we used the fact that $n \leq cm$ in the second equality. Also, in the last step, we use that $\zeta/(\zeta-2)\cdot(\zeta/2-1) = \zeta/(\zeta-2)\cdot(\zeta-2)/2 = \zeta/2$. Recall that our goal is to show that the above probability is bounded by $2^{-\Omega(\zeta x)}$. Therefore, it is sufficient to pick $x \leq m/(2\cdot\zeta^{\zeta/(\zeta-2)}) = m/(2\cdot\zeta^{1+O(1/\zeta)})$. Note that $\zeta^{1/\zeta} = 2^{\log\zeta/\zeta} = O(1)$. So, we can pick some constant $0 < \beta < 1$ so that $x \leq \beta\cdot m/\zeta$ such that the above probability is at most $2^{-\Omega(\zeta x)}$ as required. $\square$

To summarize, if $\boldsymbol{M}$ is generated by $\mathcal{M}_{\zeta\mathbf{col}}$, we show that $\boldsymbol{M}$ will not contain any null space vectors of size $c_x \leq x \leq \beta\cdot m/\zeta$ for some choice of $c_x = \Omega(\lambda+\log m)$ and constant $0 < \beta < 1$. Recall that our goal was to prove that any sub-matrix, $\boldsymbol{M}^C$, consisting of $h = \eta\cdot m$ columns $C \subseteq [n]$ of $\boldsymbol{M}$ has high rank. The above shows that, for any choice of $h$ columns, the sub-matrix $\boldsymbol{M}^C$ also has no null space vectors of size $c_x \leq x \leq \beta\cdot m/\zeta$. In particular, if $\boldsymbol{M}^C$ has a null space vector of size $x$, it is easy to see that it would also be a null space vector of the transpose of the originally generated matrix $\boldsymbol{M}$.

**Step 3: Large Null Space Vectors.** Next, we will consider the case of large null space vectors of size $x > \beta\cdot m/\zeta$. Note, the prior argument cannot scale to such large null space vectors with many columns. For example, if we consider $O(m)$ columns, it is likely that all $m$ rows will be picked by at least two columns. So, the prior argument is not sufficient for large null space vectors.

Instead, we will use the following observation. Suppose that $\mathcal{M}_{\zeta\mathbf{col}}$ generates a random $m\times n$ matrix $\boldsymbol{M}$. Consider any subset of $x$ columns $C = \{i_1,\ldots,i_x\} \subseteq [n]$ and pick any arbitrary non-zero coefficients $a_1,\ldots,a_x \in \mathbb{F}\setminus\{0\}$. Let $\boldsymbol{M}^C$ be the sub-matrix corresponding to the columns $C$ and suppose there are $t$ non-zero rows in $\boldsymbol{M}^C$. Furthermore, suppose that each of the $t$ rows contains at least two non-zero entries. We can consider the probability that $C$ and $a_1,\ldots,a_x$ will be a null space vector over the random choices of the generation of $\boldsymbol{M}$. Suppose the $j$-th row is a non-zero row vector, then we can consider the $j$-th entry of the linear combination:

$$a_1\boldsymbol{M}^{i_1} + a_2\boldsymbol{M}^{i_2} + \ldots + a_x\boldsymbol{M}^{i_x}.$$

It is not hard to see that, since $a_1,\ldots,a_x$ are non-zero and there are at least two non-zero entries in the $j$-th row, the probability that the $j$-th entry of the linear combination is 0 should be small. At first, one would guess that this probability would be exactly $1/(|\mathbb{F}|-1)$. Surprisingly, we note that this probability changes and depends on the value of $x$ (that is, the number of summands). As an example, consider $|\mathbb{F}|=3$. For $x=2$, the probability can easily be seen as $1/(|\mathbb{F}|-1)=1/2$. When $x=3$, we note the probability becomes $1/4$.[7] For $x=4$, the probability becomes $3/8$. In other words, the probability changes with $x$ and can either increase or decrease. Nevertheless, we prove the following lemma formalizing the above and showing the probability is never larger than $1/(|\mathbb{F}|-1)$:

**Lemma 3.** *Consider any finite field $|\mathbb{F}| \geq 3$. For any $y \geq 1$, consider $y$ random variables $X_1, \ldots, X_y$ where each $X_i$ are drawn uniformly at random from $\mathbb{F} \setminus \{0\}$. Then, $\Pr[X_1 + \ldots + X_y = 0] \leq 1/(|\mathbb{F}|-1)$.*

*Proof.* The probability is clearly zero for $y=1$. Pick any $y \geq 2$ and consider the probability distribution of the sum of $y-1$ random variables. We can see that

$$\begin{aligned}\Pr[X_1 + \ldots + X_y = 0] &= \sum_{i \in \mathbb{F}\setminus\{0\}} \Pr[X_y = |\mathbb{F}| - i] \cdot \Pr[X_1 + \ldots X_{y-1} = i] \\ &= \frac{1}{|\mathbb{F}|-1} \sum_{i \in \mathbb{F}\setminus\{0\}} \Pr[X_1 + \ldots X_{y-1} = i] \\ &= \frac{\Pr[X_1 + \ldots X_{y-1} \neq 0]}{|\mathbb{F}|-1} \\ &\leq \frac{1}{|\mathbb{F}|-1}\end{aligned}$$

completing the proof. □

Indeed, the above is not true for $|\mathbb{F}|=2$ as the sum of any even number of non-zero entries in such fields always sums to zero. In Appendix B, we show our results do extend to fields of size $|\mathbb{F}|=2$. Now, we use the above to prove a result about null space vectors of large size:

**Lemma 4.** *Pick any constant $0 < \eta < 1$. Suppose that $\mathbb{F}$ is a finite field. Let $\boldsymbol{M}$ be a $m \times n$ matrix generated by $\mathcal{M}_{\zeta\mathbf{col}}$. Consider any subset of columns $C = \{i_1, \ldots, i_x\} \subseteq [n]$ such that $\beta m/\zeta < x \leq \eta \cdot m$ and any set of non-zero coefficients $a_1, \ldots, a_x \in \mathbb{F} \setminus \{0\}$. Then, the probability that $C$ and $a_1, \ldots, a_x$ is a null space vector is at most $(|\mathbb{F}|-1)^{-\nu x} + 2^{-\Omega(\zeta x)}$ for some constant $\nu > 1$.*

*Proof.* First, pick any constant $\nu > 1$ satisfying $\nu x < m$. Note, this is possible since $x \leq \eta \cdot m$ and $\eta$ is a constant strictly smaller than 1. Therefore, some choice of $\nu > 1$ always exists. We split the analysis into two parts. First, we will consider the probability conditioned on $\boldsymbol{M}^C$ containing $t \geq \nu x$ non-zero rows. Afterwards, we show that the probability there are less than $\nu x$ non-zero rows is also small.

Suppose that $\boldsymbol{M}^C$ has $t \geq \nu x$ non-zero row vectors. Note, if any of the $t$ non-zero row vectors contains exactly one non-zero entry, then it is impossible for

[7] Indeed, the only possibilities for terms $a_1\boldsymbol{M}^{i_1}, a_2\boldsymbol{M}^{i_2}, a_3\boldsymbol{M}^{i_3}$ to sum to 0 in $\mathbb{F}_3$ are $(1,1,1)$ and $(2,2,2)$, out of 8 total (equally likely) possibilities.

$C$ and $a_1, \ldots, a_x$ to be a null space vector. Suppose that each of the $t$ non-zero row vectors contains at least two non-zero entries. Suppose the $j$-th row of $\boldsymbol{M}^C$ is a non-zero row vector with $y \geq 2$ non-zero entries that are chosen uniformly at random from $\mathbb{F} \setminus \{0\}$. Consider the $j$-th entry of the linear combination:

$$a_1 \boldsymbol{M}^{i_1} + a_2 \boldsymbol{M}^{i_2} + \ldots + a_x \boldsymbol{M}^{i_x}.$$

By Lemma 3, the probability that $y \geq 2$ uniformly random non-zero elements in $\mathbb{F}$ sum to zero is at most $1/(|\mathbb{F}| - 1)$. Note, the elements in each row are chosen independently at random. So, the probability that all $t \geq \nu x$ entries of the linear combination (corresponding to the $t$ non-zero row vectors) will be zero is $(|\mathbb{F}| - 1)^{-t} \leq (|\mathbb{F}| - 1)^{-\nu x}$ as $t \geq \nu x$.

Next, we consider the probability that the conditioned event of at least $\nu x$ non-zero row vectors is false. Fix any subset of $t < \nu x$ rows and we will calculate the probability that any subset of $x$ columns will only pick one of the $t$ rows in the fixed subset. A single column will pick these $t$ rows with probability $(t/m)^{\zeta}$. So, the probability that all $x$ columns will do this is $(t/m)^{\zeta x}$. Finally, we perform a Union bound over all $\binom{m}{t}$ possible subsets to get

$$\binom{m}{t} \left(\frac{t}{m}\right)^{\zeta x} \leq \left(\frac{em}{t}\right)^{t} \left(\frac{t}{m}\right)^{\zeta x} \leq \left(\frac{em}{\nu x}\right)^{\nu x} \left(\frac{\nu x}{m}\right)^{\zeta x}$$

where we applied the upper bound $(a/b)^b \leq \binom{a}{b}$ and the fact that the probability is maximized by setting $t = \nu x$. By our choice of $\nu$, we know that $\nu x/m < 1$. Therefore, we know that $(\nu x/m)^{\zeta x} = 2^{-\Omega(\zeta x)}$. On the other hand, we know that $x = \Omega(m/\zeta)$, so we know that $\left(\frac{em}{\nu x}\right)^{\nu x} = O(\zeta)^{\nu x} = 2^{O(x \log \zeta)}$. Therefore, we see that this probability is at most $2^{O(x \log \zeta)} \cdot 2^{-\Omega(\zeta x)} = 2^{-\Omega(x(\zeta - \log \zeta))} = 2^{-\Omega(\zeta x)}$. Finally, the probability that $C$ and $a_1, \ldots, a_x$ is a null space vector is at most $(|\mathbb{F}| - 1)^{-\nu x} + 2^{-\Omega(\zeta x)}$ for some constant $\nu > 1$. $\square$

The above lemma considers the probability that a single choice of subset of columns $C$ and non-zero coefficients will be a null space vector. We show that this is sufficient to show that any matrix generated by our matrix family $\mathcal{M}_{\zeta\mathbf{col}}$ will also not contain null space vectors of size $x$ for any $x = \Omega(m/\zeta)$.

**Lemma 5.** *Pick any constant $c \geq 1$ as well as any finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq 3$ and parameter $\zeta = \Theta(\log |\mathbb{F}|)$. Let $n = c \cdot m$ and suppose $\boldsymbol{M}$ is a $m \times n$ matrix generated by $\mathcal{M}_{\zeta\mathbf{col}}$. Then, for every constant $0 < \eta < 1$ and $0 < \beta < 1$, the probability there exists at least one null space vector of any size $\beta \cdot m/\zeta < x \leq \eta \cdot m$ in $\boldsymbol{M}$ is at most $2^{-2\lambda}$ for sufficiently large $m = \Omega(\lambda)$.*

*Proof.* Consider any fixed size $\beta \cdot m/\zeta \leq x \leq \eta \cdot m$ for null space vectors. Let $Y^x$ be the random variable denoting the number of null space vectors of size $x$ in $\boldsymbol{M}$. Now, fix any subset of $x$ columns denoted by $C = \{i_1, \ldots, i_x\} \subseteq [n]$ as well as $x$ non-zero coefficients $a_1, \ldots, a_x$. By Lemma 4, we know that $C$ and $a_1, \ldots, a_x$ is a null space vector with probability at most $(|\mathbb{F}| - 1)^{-\nu x} + 2^{-\Omega(\zeta x)}$ for some constant $\nu > 1$. Next, we can apply a Union bound over all choices of columns and non-zero coefficients to obtain a bound on the number of null space vectors of size $x$ in $\boldsymbol{M}$ denoted by $Y^x$. The number of choices of columns is $\binom{n}{x} = \binom{cm}{x}$ and the number of choices of non-zero coefficients is $(|\mathbb{F}|-1)^x$. So, we get

$$\begin{aligned}
\Pr[Y^x \geq 1] &\leq \binom{cm}{x} \cdot (|\mathbb{F}|-1)^x \cdot ((|\mathbb{F}|-1)^{-\nu x} + 2^{-\Omega(\zeta \cdot x)}) \\
&\leq 2^{O(x \cdot \log \zeta)} \cdot ((|\mathbb{F}|-1)^{-(\nu-1)x} + 2^{-\Omega(x(\zeta - \log|\mathbb{F}|))}) \\
&\leq 2^{O(x \cdot \log \zeta)} \cdot (2^{-\Omega(\log(|\mathbb{F}|-1)x)} + 2^{-\Omega(\zeta x)}) \\
&\leq 2^{O(x \cdot \log \zeta)} \cdot 2^{-\Omega(\zeta x)} \\
&= 2^{-\Omega(\zeta x)}.
\end{aligned}$$

In the second inequality, we used that $x = \Omega(m/\zeta)$ to see that $\binom{cm}{x} = 2^{O(x \log \zeta)}$. In the third inequality, we use the fact that $\nu$ is a constant such that $\nu > 1$ and assume that $\zeta = \Theta(\log|\mathbb{F}|)$. In the fourth inequality, we use that $\zeta = \Theta(\log|\mathbb{F}|)$. We get that the above probability is at most $\Pr[Y^x \geq 1] \leq 2^{-\Omega(\zeta x)} \leq 2^{-2\lambda - \log m}$ as $x \geq \beta \cdot m/\zeta$ and we assume $m = \Omega(\lambda)$. Finally, we apply a Union bound over all $O(m)$ choices of null space vector sizes $x$ to get that

$$\sum_{x=\beta \cdot m/\zeta}^{\alpha \cdot m} \Pr[Y^x \geq 1] \leq m \cdot 2^{-2\lambda - \log m} = 2^{-2\lambda}$$

to complete the proof. □

**Step 4: Putting it All Together.** Finally, we utilize all the results from the prior three steps to prove that $\mathcal{M}_{\zeta\mathbf{col}}$ is a weakly super-invertible matrix family of dimension $m \times n$ where the number of non-zero entries in each columns is $\zeta = \Theta(\log|\mathbb{F}|)$. One can choose any constant $c \geq 1$ such that the number of columns is $n = c \cdot m$ and any constants $0 < \gamma < 1$ and $\gamma < \eta \leq c$ such that every sub-matrix of $\eta \cdot m$ columns will always have rank at least $\gamma \cdot m$ except with probability exponentially small in $\lambda$. We will now prove that this is true:

*Proof of Theorem 4.* Let $\boldsymbol{M}$ be a $m \times n$ matrix generated randomly by $\mathcal{M}_{\zeta\mathbf{col}}$. Now, consider any subset of $\eta \cdot m$ columns of $\boldsymbol{M}$ denoted by $C \subseteq [n]$. Recall that our goal is to show that the corresponding sub-matrix $\boldsymbol{M}^C$ has rank at least $\gamma \cdot m$ for some chosen constant $\gamma < 1$. We show that the sub-matrix $\boldsymbol{M}^C$ has rank at least $|C| - O(\lambda \log m + \log^2 m) = \eta \cdot m - O(\lambda \log m + \log^2 m) \geq \gamma \cdot m$ that follows as $\eta > \gamma$ and the fact that $m = \Omega(\lambda \log m)$ is sufficiently large.

It remains to show that $\boldsymbol{M}^C$ has large rank. First, by combining Lemma 2 and Lemma 5, we know that $\boldsymbol{M}$ has no null space vectors of size $x$ where $c_x \leq x \leq \eta \cdot m$ for any constant $0 < \eta < 1$ and some $c_x = \Omega(\lambda + \log m)$ except with probability $2 \cdot 2^{-2\lambda} \leq 2^{-\lambda}$. Thus, $\boldsymbol{M}^C$ has no null space vectors of size more than $c_x$ as $\boldsymbol{M}^C$ is a sub-matrix of $\boldsymbol{M}$ with $\eta \cdot m$ columns. Therefore, the total number of null space vectors in $\boldsymbol{M}^C$ is at most $\sum_{x=0}^{c_x - 1} Y^x \leq \sum_{x=0}^{c_x-1} \binom{n}{x} \cdot (|\mathbb{F}|-1)^x \leq c_x \cdot |\mathbb{F}|^{c_x} \cdot \binom{n}{c_x} \leq c_x \cdot (n \cdot |\mathbb{F}|)^{c_x}$ as we know that there are no null space vectors of size $c_x$ or larger. Next, we apply Lemma 1 to see that the rank of the sub-matrix $\boldsymbol{M}^C$ is at least

$$\eta \cdot m - \log_{|\mathbb{F}|}(c_x \cdot (n \cdot |\mathbb{F}|)^{c_x}) \geq \eta \cdot m - c_x(\log_{|\mathbb{F}|} n + 1) - \log_{|\mathbb{F}|} c_x$$

as the number of columns was $|C| = \eta \cdot m$. Suppose $m$ is chosen to satisfy

$$m \geq (c_x(\log_{|\mathbb{F}|} n + 1) + \log_{|\mathbb{F}|} c_x)/(\eta - \gamma) = \Omega((\lambda + \log m) \log m).$$

Note, it suffices to choose $m = \Omega(\lambda \log m)$ as $m = \Omega(\log^2 m)$ is already true for sufficiently large constant $m$. Then, we can see that the rank of the sub-matrix $\boldsymbol{M}^C$ is at least $\eta \cdot m - c_x(\log_{|\mathbb{F}|} n + 1) - \log_{|\mathbb{F}|} c_x \geq \gamma \cdot m$, except with probability $2^{-\lambda}$ for any subset of $\eta \cdot m$ columns $C \subseteq [n]$, completing the proof. □

*Optimality of $\mathcal{M}_{\zeta\textbf{col}}$.* Our construction $\mathcal{M}_{\zeta\textbf{col}}$ is asymptotically optimal in terms of $L_0$ norm as it is easy to see that each row vector of any weakly super-invertible matrix must have at least $\Omega(1)$ $L_0$ norm in expectation. This can be easily be seen by picking any $m \times h$ sub-matrix where $h = \Omega(m)$ and noting that this sub-matrix must have rank at least $\gamma \cdot m = \Omega(m)$. So, $\Omega(m)$ rows must have at least one non-zero entry. Our construction $\mathcal{M}_{\zeta\textbf{col}}$ with $|\mathbb{F}| = 3$ and $\zeta = O(1)$ has $O(m)$ non-zero entries matching this lower bound. Although, we note that our MPC applications utilize $\mathcal{M}_{\zeta\textbf{col}}$ with larger fields $|\mathbb{F}| = O(n + 2^\lambda)$ that have $O(m \cdot (\lambda + \log n))$ non-zero entries that is optimal up to $\lambda$ and $\log n$ factors.

*Weakly Hyper-Invertible Matrices.* One could also try to weaken the requirements for hyper-invertibility to obtain sparse matrices. In Appendix B.3, we show that a similar weakening for hyper-invertible matrices still requires $\Omega(n)$ $L_0$ norm for each row. In other words, there must be $\Omega(n \cdot m)$ non-zero entries in the entire matrix. So, Vandermonde matrices have optimal sparsity. Therefore, it does not make sense to study the weakening for hyper-invertibility.

## 6 Instantiating the Preprocessing of [GPS22]

As stated earlier, the ultimate goal of our paper is to instantiate the preprocessing of [GPS22, Functionality 15] with $\widetilde{O}(|C|)$ communication and adaptive security, in the $t < (1 - \varepsilon)n$ setting. Assuming this preprocessing, [GPS22] obtain an MPC protocol with $O(|C|)$ online communication in this setting. In this section, we present the preprocessing functionality $\mathcal{F}_{\textsf{prep-mal}}$ and our protocol with $\widetilde{O}(|C|)$ communication.

### 6.1 The Preprocessing Functionality

We now present the preprocessing Functionality verbatim from [GPS22]. The functionality begins by generating a random $\gamma \leftarrow_\$ \mathbb{F}$ and sets $\boldsymbol{\gamma} \leftarrow (\gamma, \gamma, \ldots, \gamma)^\intercal \in \mathbb{F}^k$. This $\gamma$ serves as the MAC key in the online phase, and ensures that shared values are opened correctly. Indeed, in the online phase, all circuit values will be shared using degree-$(n-k)$ *authenticated* packed Shamir sharings $[\![\boldsymbol{v}]\!]_{n-k} := ([\boldsymbol{v}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{v}]_{n-k})$, where the latter sharing is referred to as the 'MAC' on the former sharing. Intuitively, if $\boldsymbol{v}$ is opened then it will be checked by opening $\boldsymbol{\gamma} * \boldsymbol{v}$—since $\gamma$ is random, with all-but-negligible probability, the adversary will be unable to force an opening to some $\boldsymbol{v}' \neq \boldsymbol{v}$.[8] $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ then has two procedures:

1. $\mathsf{rand\text{-}sharing}(\boldsymbol{r}, d)$: Samples a random degree-$d$ packed Shamir sharing $[\boldsymbol{r}]_d$ that is consistent with shares input by the adversary for corrupt parties, then sends the honest parties their shares.
2. $\mathsf{auth\text{-}sharing}(\boldsymbol{r})$: Samples degree-$(n-k)$ authenticated packed Shamir sharings $[\![\boldsymbol{r}]\!]_{n-k}$ consistent with input corrupt parties' shares as above, then sends the honest parties their shares.

$\mathcal{F}_{\mathsf{prep\text{-}mal}}$ uses the procedures as follows. First, it uses rand-sharing to prepare $[\boldsymbol{\gamma}]_{n-k}$. Then, for ever group of $k$ input and output gates, it uses auth-sharing to prepare $[\![\boldsymbol{r}]\!]_{n-k}$ for random $\boldsymbol{r} \in \mathbb{F}^k$, then rand-sharing to prepare $[\boldsymbol{\Delta}]_{n-k}$ and $[\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k}$ for random $\Delta \in \mathbb{F}^k$, and finally rand-sharing to prepare random $[\boldsymbol{o}]_{n-1}$, where $\boldsymbol{o} = \boldsymbol{0}$. The random sharings are used to securely share and reconstruct inputs and outputs, respectively, in the presence of a malicious adversary.

For each group of $k$ multiplication gates, $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ prepares packed beaver triples as follows. It samples two random $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$, computes $\boldsymbol{c} \leftarrow \boldsymbol{a} * \boldsymbol{b}$, and finally inputs them to auth-sharing to prepare $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$. It also prepares random sharings $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$, where $\boldsymbol{o}^{(1)} = \boldsymbol{o}^{(2)} = \boldsymbol{0}$, using rand-sharing.

Finally, for a verification phase at the end of the online phase which is used to catch a cheating adversary, $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ prepares additional random sharings $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$, where $\boldsymbol{o}^{(1)} = \boldsymbol{o}^{(2)} = \boldsymbol{0}$, using rand-sharing. It also samples random $r \in \mathbb{F}$, computes $\gamma \cdot r$, and generates a pair of additive sharings $(\langle r \rangle, \langle \gamma \cdot r \rangle)$, where the shares for corrupted parties are input by the adversary, then distributes to the honest parties their shares of these sharings.

**Functionality 1: $\mathcal{F}_{\mathsf{prep\text{-}mal}}$**

$\mathcal{F}_{\mathsf{prep\text{-}mal}}$ receives the set of corrupted parties, denoted by $\mathsf{Corr}$ $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples a random field element $\gamma \in \mathbb{F}$ and sets $\boldsymbol{\gamma} \leftarrow (\gamma, \gamma, \ldots, \gamma) \in \mathbb{F}^k$. Let $d \in \{n-k, n-1\}$. We define the following two procedures.

– $\mathsf{rand\text{-}sharing}(\boldsymbol{r}, d)$: $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ receives from the adversary a set of shares $\{r_j\}_{j \in \mathsf{Corr}}$. Then $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples a random degree-$d$ packed Shamir sharing $[\boldsymbol{r}]_d$ such that for all $P_j \in \mathsf{Corr}$, the $j$-th share of $[\boldsymbol{r}]_d$ is $r_j$. Finally, $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ distributes the shares of $[\boldsymbol{r}]_d$ to honest parties.
– $\mathsf{auth\text{-}sharing}(\boldsymbol{r})$: $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ receives from the adversary a set of shares $\{(r_j, u_j)\}_{j \in \mathsf{Corr}}$. Then $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ computes two degree-$(n-k)$ packed Shamir sharings $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$ such that for all $P_j \in \mathsf{Corr}$, the $j$-th shares of $([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$ are $r_j, u_j$, respectively. Finally $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ distributes the shares of $([\![\boldsymbol{r}]\!]_{n-k} = ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k})$ to honest parties.

The ideal functionality $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ runs the following steps.

[8] This is done once for many such openings at a time using random linear combinations.

1. $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ invokes $\mathsf{rand\text{-}sharing}(\boldsymbol{\gamma}, n-k)$ to prepare $[\boldsymbol{\gamma}]_{n-k}$.
2. For every group of $k$ input gates and output gates:
   (a) $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples a random vector $\boldsymbol{r} \in \mathbb{F}^k$ and invokes $\mathsf{auth\text{-}sharing}(\boldsymbol{r})$ to prepare $[\![\boldsymbol{r}]\!]_{n-k}$.
   (b) $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples a random vector $\boldsymbol{\Delta} \in \mathbb{F}^k$ and invokes $\mathsf{rand\text{-}sharing}(\boldsymbol{\Delta}, n-k)$ and $\mathsf{rand\text{-}sharing}(\boldsymbol{\Delta} * \boldsymbol{r}, n-k)$ to prepare $([\boldsymbol{\Delta}]_{n-k}, [\boldsymbol{\Delta} * \boldsymbol{r}]_{n-k})$.
   (c) For every group of $k$ output gates, $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ invokes $\mathsf{rand\text{-}sharing}(\mathbf{0}, n-1)$ to prepare $[\boldsymbol{o}]_{n-1}$, where $\boldsymbol{o} \leftarrow \mathbf{0}$.
3. For every group of $k$ multiplication gates:
   (a) $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples two random vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$ and computes $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Then, $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ invokes $\mathsf{auth\text{-}sharing}(\boldsymbol{a})$, $\mathsf{auth\text{-}sharing}(\boldsymbol{b})$, and $\mathsf{auth\text{-}sharing}(\boldsymbol{c})$ to prepare $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$.
   (b) $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ invokes two times of $\mathsf{rand\text{-}sharing}(\mathbf{0}, n-1)$ to prepare $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$, where $\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)} \leftarrow \mathbf{0}$.
4. All parties prepare the following random sharings for the verification of the computation:
   (a) All parties invoke two times of $\mathsf{rand\text{-}sharing}(\mathbf{0}, n-1)$ to prepare $[\boldsymbol{o}^{(1)}]_{n-1}, [\boldsymbol{o}^{(2)}]_{n-1}$, where $\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)} \leftarrow \mathbf{0}$.
   (b) $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ receives from the adversary a set of shares $\{(r_j, r'_j)\}_{j \in \mathsf{Corr}}$. Then $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples a random field element $r$ and computes $\gamma \cdot r$. $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ randomly generates a pair of additive sharings $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ such that for all $P_j \in \mathsf{Corr}$, the $j$-th shares of $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ are $r_j, r'_j$, respectively. Finally, $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ distributes the shares of $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ to honest parties.

## 6.2 Useful Sub-Functionalities and Sub-Procedures

We now describe important Sub-Functionalities and Sub-Procedures for $\Pi_{\mathsf{prep\text{-}mal}}$.

**Oblivious Linear Evaluation.** The first functionality, $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$, is a two party functionality that generates several random correlations between the two parties, known as oblivious linear evaluation (OLE). More specifically, the two parties $P_A$ and $P_B$ input $\boldsymbol{u} \in \mathbb{F}^t$ and $\boldsymbol{x} \in \mathbb{F}^t$, respectively, to $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$, which samples random $\boldsymbol{v} \leftarrow_{\$} \mathbb{F}^t$, and outputs $\boldsymbol{w} \leftarrow \boldsymbol{u} * \boldsymbol{x} - \boldsymbol{v}$ to $P_A$ and $\boldsymbol{v}$ to $P_B$. Observe that $\boldsymbol{x}$ remains random to $P_A$ and $\boldsymbol{u}$ remains random to $P_B$. We will assume that $t \geq k$.

**Functionality 2: $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$**

**Parameters**: The functionality runs between parties $P_A$ and $P_B$.

On receiving $\boldsymbol{u} \in \mathbb{F}^t$ from $P_A$ and $\boldsymbol{x} \in \mathbb{F}^t$ from $P_B$, sample $\boldsymbol{v} \leftarrow_{\$} \mathbb{F}_p^t$ then output $\boldsymbol{w} \leftarrow \boldsymbol{u} * \boldsymbol{x} - \boldsymbol{v}$ to $P_A$ and $\boldsymbol{v}$ to $P_B$.

**Corrupted Party**: If $P_B$ is corrupted, $\boldsymbol{v}$ may be chosen by $\mathcal{A}$. If $P_A$ is corrupted, $\boldsymbol{w}$ can be chosen by $\mathcal{A}$ (and $\boldsymbol{v}$ is computed by $\boldsymbol{u} * \boldsymbol{x} - \boldsymbol{w}$).

We assume the existence of a protocol that instantiates $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$. A number of such protocols exist [BCG+19b, BCG+20, BCGI18, WYKW21, BCG+19a, BDOZ11, DPSZ12, DKL$^+$13]. Although none of the above protocols specifically built for $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ prove adaptive security, we note that the generic adaptively- and maliciously-secure 2PC protocol from [CLOS02] can be used to realize each underlying OLE correlation with $\mathtt{poly}(\kappa)$ communication complexity, which will suffice for our purposes. We refer to the efficiency of creating $t$ OLE correlations from $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ as $|\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}|$. All other functionalities in this paper, we will directly instantiate (perhaps using $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$).

**Omitted Functionalities and Procedures.** We present and instantiate the following standard functionalities and procedures in Appendix C:

- $\mathcal{F}_{\mathsf{verify\text{-}deg}}$: This functionality takes in several packed Shamir sharings which are supposed to be of degree-$(k-1)$. If any are not, it sends $\mathsf{abort}$ to the honest parties. This is accomplished in the usual way, by taking a random linear combination of the sharings, and then opening the resulting sharing to all parties, who check its degree.
- $\mathcal{F}_{\mathsf{rand}}$: This functionality generates random sharings (under some secret sharing scheme, which will be clear from context) of uniformly random values. This is accomplished using the standard technique of applying a super-invertible matrix for randomness extraction [DN07].
- $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$: This functionality takes as input sharings $[\boldsymbol{u}_i]_{n-1}$ and outputs $[\boldsymbol{u}_i]_{n-k}$. This is accomplished by using the standard technique of masking it with a random value $[\boldsymbol{r}_i]_{n-1}$, opening, and unmasking with $[\boldsymbol{r}_i]_{n-k}$. These random pairs are generated using $\mathcal{F}_{\mathsf{rand}}$.
- $\pi_{\mathsf{sacrifice}}$: This procedure takes as input two authenticated packed beaver triples and outputs the first if it has no error. This is done using the standard triple sacrificing procedure of [DPSZ12], which uses a random challenge to ensure that there is no error.
- $\pi_{\mathsf{check\text{-}zero}}$: In this procedure, the parties take as input several random sharings $[\boldsymbol{\theta}_i]_{n-1}$ that are supposed to satisfy $\boldsymbol{\theta}_i = \mathbf{0}$, and abort if any do not. This procedure uses the standard technique of taking a random linear combination of the sharings, having parties commit to their shares, and then having parties open their shares to each other and check that the underlying secret is $\mathbf{0}$.

## 6.3 Packed Beaver Triples from Weakly Super-Invertible Matrices

We present our main contribution to instantiate $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ for generating random packed beaver triples. First, we present the ideal functionality $\mathcal{F}_{\mathsf{triple}}$. Let $m \in \mathbb{N}, 1/2 < \gamma < 1$, $\mu \leftarrow \gamma \cdot m - (m+1)/2$, and $t \geq k$. $\mathcal{F}_{\mathsf{triple}}$ generates random packed triple sharings with error, $([\boldsymbol{a}_{l,w}]_{n-k}, [\boldsymbol{b}_{l,w}]_{n-k}, [\boldsymbol{a}_{l,w} * \boldsymbol{b}_{1,w} + \boldsymbol{\delta}_{1,w}]_{n-1})$, for $l \in [\mu], w \in [\lfloor t/k \rfloor]$, where the errors $\boldsymbol{\delta}_{1,w}, \ldots, \boldsymbol{\delta}_{\mu,w}$ are input by the adversary. We explain the need to allow the adversary to input such errors below. These sharings will be consistent with shares for corrupted parties input by the adversary.

**Functionality 3: $\mathcal{F}_{\mathsf{triple}}$**

**Parameters**: Let $m \in \mathbb{N}, 1/2 < \gamma < 1$, and $\mu \leftarrow \gamma \cdot m - (m+1)/2$.

1. $\mathcal{F}_{\mathsf{triple}}$ receives from the adversary the corrupted parties' shares, $\{(a^j_{1,w}, b^j_{1,w}, c^j_{1,w}), \ldots, (a^j_{\mu,w}, b^j_{\mu,w}, c^j_{\mu,w})\}_{j \in \mathsf{Corr}, w \in [\lfloor t/k \rfloor]}$, and errors $(\boldsymbol{\delta}_{1,w}, \ldots, \boldsymbol{\delta}_{\mu,w})_{w \in [\lfloor t/k \rfloor]}$.
2. $\mathcal{F}_{\mathsf{triple}}$ samples random $((\boldsymbol{a}_{1,w}, \boldsymbol{b}_{1,w}), \ldots, (\boldsymbol{a}_{\mu,w}, \boldsymbol{b}_{\mu,w}))_{w \in [\lfloor t/k \rfloor]}$.
3. $\mathcal{F}_{\mathsf{triple}}$ samples random sharings $(([\boldsymbol{a}_{1,w}]_{n-k}, [\boldsymbol{b}_{1,w}]_{n-k}, [\boldsymbol{a}_{1,w} * \boldsymbol{b}_{1,w} + \boldsymbol{\delta}_{1,w}]_{n-1}), \ldots, ([\boldsymbol{a}_{\mu,w}]_{n-k}, [\boldsymbol{b}_{\mu,w}]_{n-k}, [\boldsymbol{a}_{\mu,w} * \boldsymbol{b}_{\mu,w} + \boldsymbol{\delta}_{\mu,w}]_{n-1}))_{w \in [\lfloor t/k \rfloor]}$ such that for all $j \in \mathsf{Corr}, w \in [\lfloor t/k \rfloor]$, the $j$-th shares are $(a^j_{1,w}, b^j_{1,w}, c^j_{1,w}), \ldots, (a^j_{\mu,w}, b^j_{\mu,w}, c^j_{\mu,w})$.
4. $\mathcal{F}_{\mathsf{triple}}$ then distributes to the honest parties their shares.

We now present our protocol $\Pi_{\mathsf{triple}}$. To generate the random packed triple sharings, we will need to perform beaver multiplication (using packed triples that may not be uniformly random). We do this in the standard procedure $\pi_{\mathsf{beaver}}$ below. Since $P_1$ distributes degree-$(k-1)$ packed Shamir sharings, we will need to use $\Pi_{\mathsf{verify\text{-}deg}}$ to check their degrees whenever we use $\pi_{\mathsf{beaver}}$, such as in $\Pi_{\mathsf{triple}}$.

**Procedure 1: $\pi_{\mathsf{beaver}}$**

1. All parties hold shares of a packed beaver triple $([\boldsymbol{a}]_{n-k}, [\boldsymbol{b}]_{n-k}, [\boldsymbol{c}]_{n-k})$ and shares of packed inputs $([\boldsymbol{x}]_{n-k}, [\boldsymbol{y}]_{n-k})$.
2. The parties first locally compute $[\boldsymbol{x} - \boldsymbol{a}]_{n-k}$ and $[\boldsymbol{y} - \boldsymbol{b}]_{n-k}$ and open them to $P_1$.
3. $P_1$ then reconstructs $\boldsymbol{d} \leftarrow \boldsymbol{x} - \boldsymbol{a}$, $\boldsymbol{e} \leftarrow \boldsymbol{y} - \boldsymbol{b}$ and computes then distributes sharings $[\boldsymbol{d}]_{k-1}$, $[\boldsymbol{e}]_{k-1}$.
4. Finally, the parties locally compute $[\boldsymbol{x} * \boldsymbol{y}]_{n-1} \leftarrow [\boldsymbol{d}]_{k-1} * [\boldsymbol{e}]_{k-1} + [\boldsymbol{d}]_{k-1} * [\boldsymbol{b}]_{n-k} + [\boldsymbol{e}]_{k-1} * [\boldsymbol{a}]_{n-k} + [\boldsymbol{c}]_{n-k}$.

Now we can present $\Pi_{\mathsf{triple}}$ that is parameterized by a family of $(\delta, \gamma, \eta, \ell)$-weakly super-invertible $m \times n$ matrices, $\mathcal{F} = (\mathcal{S}, \mathsf{Gen})$, where $\delta = \mathtt{negl}(\lambda)$, $1/2 < \gamma < 1$, $2\varepsilon n > m = \Omega(n)$, $1/(2\varepsilon) > \eta = n - t > \varepsilon \cdot n$, and $\ell = O(\lambda)$. This means that with-all-but-negligible probability, any matrix $\boldsymbol{M}$ sampled by $\mathsf{Gen}$ will be such that every $m \times \varepsilon n$ sub-matrix of $\boldsymbol{M}$ will have rank more than $m/2$.

As discussed in Section 2, $\Pi_{\mathsf{triple}}$ proceeds via two 'levels' of extraction. We overview these two levels and point to Section 2 for a more detailed description.

*First level of extraction.* In the first level, each party $P_i$ samples random $\boldsymbol{u}_i, \boldsymbol{v}_i \in \mathbb{F}^t$, and then splits them into $\lfloor t/k \rfloor$ length-$k$ vectors $\boldsymbol{u}_{i,w}$ and $\boldsymbol{v}_{i,w}$. Next, the parties jointly sample a weakly super-invertible matrix $\boldsymbol{M}$ using $\mathsf{Gen}$ on input randomness from $\mathcal{F}_{\mathsf{coin}}$. Then, for $w \in [\lfloor t/k \rfloor]$, each party $P_i$ distributes random packed Shamir sharings $[\boldsymbol{u}_{i,w}]_{n-k}$ and $[\boldsymbol{v}_{i,w}]_{n-k}$, and uses $\boldsymbol{M}$ to extract from these two sets of $n$ packed sharings, new sharings $[\boldsymbol{a}_{1,w}]_{n-k}, \ldots, [\boldsymbol{a}_{m,w}]_{n-k}$ and

$[\boldsymbol{b}_{1,w}]_{n-k}, \ldots, [\boldsymbol{b}_{m,w}]_{n-k}$, respectively. We have that for $l \in [m]$,

$$\begin{aligned}\boldsymbol{c}_{l,w} &:= \boldsymbol{a}_{l,w} * \boldsymbol{b}_{l,w} \\ &= \sum_{i \in [n]: \boldsymbol{M}_l[i] \neq 0} \boldsymbol{M}_l[i] \cdot \left( \sum_{j \in [n] \setminus \{i\}: \boldsymbol{M}_l[j] \neq 0} \boldsymbol{M}_l[j] \cdot (\boldsymbol{u}_{i,w} * \boldsymbol{v}_{j,w} + \boldsymbol{u}_{j,w} * \boldsymbol{v}_{i,w}) \right) \\ &\quad + \sum_{i \in [n]: \boldsymbol{M}_l[i] \neq 0} (\boldsymbol{M}_l[i])^2 \cdot \boldsymbol{u}_{i,w} * \boldsymbol{v}_{i,w}.\end{aligned}$$

Thus, in order to get the sharing $[\boldsymbol{c}_{l,w}]_{n-k}$, first, for every ordered pair $(P_i, P_j)$ of parties such that $\boldsymbol{M}_l[i], \boldsymbol{M}_l[j] \neq 0$, the parties invoke $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ on their respective seeds to obtain $\boldsymbol{\alpha}_{i,w}^j, \boldsymbol{\beta}_{j,w}^i$, respectively, such that $\boldsymbol{\alpha}_{i,w}^j + \boldsymbol{\beta}_{j,w}^i = \boldsymbol{u}_{i,w} * \boldsymbol{v}_{j,w}$. Then, for each $P_i$ in which $\boldsymbol{M}_l[i] \neq 0$, it computes

$$\boldsymbol{c}_{l,w}^i \leftarrow (\boldsymbol{M}_l[i])^2 \cdot \boldsymbol{u}_{i,w} * \boldsymbol{v}_{i,w} + \boldsymbol{M}_l[i] \cdot \sum_{j \in [n] \setminus \{i\}: \boldsymbol{M}_l[j] \neq 0} \boldsymbol{M}_l[j] \cdot (\boldsymbol{\alpha}_{i,w}^j + \boldsymbol{\beta}_{i,w}^j)$$

then computes and distributes degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{c}_{l,w}^i]_{n-k}$. Finally, all of the parties set $[\boldsymbol{c}_{l,w}]_{n-k} \leftarrow \sum_{i: \boldsymbol{M}_l[i] \neq 0} [\boldsymbol{c}_{l,w}^i]_{n-k}$.

*Second level of extraction.* Since $\boldsymbol{M}$ is only *weakly* super-invertible, the extracted values $\boldsymbol{a}_{l,w}, \boldsymbol{b}_{l,w}$ may not be fully random. Indeed, for each $w \in [\lfloor t/k \rfloor]$, we are only guaranteed that some $\gamma \cdot m$ of them are fully random, but we do not know which ones. This leads us to our second level of extraction, which is based on the "triple extraction" protocol of [CP17]. In this level of extraction, for every $w \in [\lfloor t/k \rfloor]$, we are able to extract $\mu = \gamma \cdot m - (m+1)/2$ random packed beaver triples from the $m$ packed triples output by the first level of extraction.

*Errors.* Note that there are two places where corrupted parties $P_i$ could inject errors $\boldsymbol{\delta}$ such that $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b} + \boldsymbol{\delta}$ for our output triples in this protocol:

- $P_i$ could input to $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ different seeds than those corresponding to the $[\boldsymbol{u}_{i,w}]_{n-k}, [\boldsymbol{v}_{i,w}]_{n-k}$ that they share.
- A corrupt party $P_1$ could distribute incorrect sharings $[\overline{\boldsymbol{d}}]_{k-1}, [\overline{\boldsymbol{e}}]_{k-1}$ in $\pi_{\mathsf{beaver}}$.

**Protocol 2: $\Pi_{\mathsf{triple}}$**

**Parameters**: This protocol is parameterized by a $(\delta, \gamma, \eta, \ell)$-weakly super-invertible $m \times n$ matrix, $\mathcal{F} = (\mathcal{S}, \mathsf{Gen})$, where $\delta = \mathtt{negl}(\lambda)$, $1/2 < \gamma < 1$, $2\varepsilon n > m = \Omega(n)$, $1/(2\varepsilon) > \eta = n - t > \varepsilon \cdot n$, and $\ell = O(\lambda)$. Let $\mu \leftarrow \gamma \cdot m - (m+1)/2$. We assume w.l.o.g. that $m$ is odd.

**Extraction Level 1**:

1. Each party $P_i$ first samples random $\boldsymbol{u}_i \leftarrow_\$ \mathbb{F}^t$ and $\boldsymbol{v}_i \leftarrow_\$ \mathbb{F}^t$.
2. Then all parties invoke $\mathcal{F}_{\mathsf{coin}}$ to obtain random value $r \in \mathbb{F}$, interpret it as bit string $\mathbf{R}$, and locally compute $m \times n$ matrix, $\boldsymbol{M} \leftarrow \mathsf{Gen}(\mathbf{R})$.

3. Then, for each $l \in [m]$: for every ordered pair $(P_i, P_j)$ such that $\boldsymbol{M}_l[i], \boldsymbol{M}_l[j] \neq 0$, parties $P_i$ and $P_j$ invoke $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ on input seeds $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$, respectively, and receive back $\boldsymbol{\alpha}_i^j$ and $\boldsymbol{\beta}_j^i$, respectively, such that $\boldsymbol{\alpha}_i^j + \boldsymbol{\beta}_j^i = \boldsymbol{u}_i * \boldsymbol{v}_j$.
4. Next, each party splits $\boldsymbol{u}_i$, $\boldsymbol{v}_i$ and each $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ into length-$k$ vectors $\{\boldsymbol{u}_{i,w}\}_{w\in[t/k]}$, $\{\boldsymbol{v}_{i,w}\}_{w\in[t/k]}$, $\{\boldsymbol{\alpha}_{i,w}^j\}_{w\in[t/k]}$, $\{\boldsymbol{\beta}_{i,w}^j\}_{w\in[t/k]}$, respectively.
5. Then, for $w \in [\lfloor t/k \rfloor]$:
   (a) Each party $P_i$ computes and distributes random packed Shamir sharings $[\boldsymbol{u}_{i,w}]_{n-k}$ and $[\boldsymbol{v}_{i,w}]_{n-k}$.
   (b) Next, the parties locally compute $([\boldsymbol{a}_{1,w}]_{n-k}, \dots, [\boldsymbol{a}_{m,w}]_{n-k})^\intercal \leftarrow \boldsymbol{M} \cdot ([\boldsymbol{u}_{1,w}]_{n-k}, \dots, [\boldsymbol{u}_{n,w}]_{n-k})^\intercal$ and $([\boldsymbol{b}_{1,w}]_{n-k}, \dots, [\boldsymbol{b}_{m,w}]_{n-k})^\intercal \leftarrow \boldsymbol{M} \cdot ([\boldsymbol{v}_{1,w}]_{n-k}, \dots, [\boldsymbol{v}_{n,w}]_{n-k})^\intercal$
   (c) Then for each $l \in [m]$:
      i. Each $P_i$ such that $\boldsymbol{M}_l[i] \neq 0$ computes:
      $$\boldsymbol{c}_{l,w}^i \leftarrow (\boldsymbol{M}_l[i])^2 \cdot \boldsymbol{u}_{i,w} * \boldsymbol{v}_{i,w} + \boldsymbol{M}_l[i] \cdot \sum_{j \in [n]\setminus\{i\} : \boldsymbol{M}_l[j] \neq 0} \boldsymbol{M}_l[j] \cdot (\boldsymbol{\alpha}_{i,w}^j + \boldsymbol{\beta}_{i,w}^j),$$
      then computes and distributes degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{c}_{l,w}^i]_{n-k}$.
      ii. Then, all of the parties set $[\boldsymbol{c}_{l,w}]_{n-k} \leftarrow \sum_{i:\boldsymbol{M}_l[i]\neq 0} [\boldsymbol{c}_{l,w}^i]_{n-k}$.

**Extraction Level 2**: Let $N = (m-1)/2$. The parties do the following for each $w \in [\lfloor t/k \rfloor]$:

1. For $l \in [N+1]$, $\tau \in [k]$, the parties implicitly view $a_{l,w}^\tau, b_{l,w}^\tau, c_{l,w}^\tau$ as the $l$-th evaluation points of polynomials $A_{\tau,w}(\cdot)$ of degree $N$, $B_{\tau,w}(\cdot)$ of degree $N$, $C_{\tau,w}(\cdot)$ of degree $m-1$ $(= 2N)$, respectively; i.e., $A_{\tau,w}(l) = a_{l,w}^\tau$, $B_{\tau,w}(l) = b_{l,w}^\tau$, and $C_{\tau,w}(l) = c_{l,w}^\tau$.
   Note that the $N+1$ values $a_{l,w}^\tau, b_{l,w}^\tau$ define degree $N$ polynomials $A_{\tau,w}(\cdot)$, $B_{\tau,w}(\cdot)$, respectively.
2. Letting $\boldsymbol{A}_w(\cdot) \leftarrow (A_{1,w}(\cdot), \dots, A_{k,w}(\cdot))$, $\boldsymbol{B}_w(\cdot) \leftarrow (B_{1,w}(\cdot), \dots, B_{k,w}(\cdot))$, and $\boldsymbol{C}_w(\cdot) \leftarrow (C_{1,w}(\cdot), \dots, C_{k,w}(\cdot))$, the parties have
   $$([\boldsymbol{A}_w(1)]_{n-k}, \dots, [\boldsymbol{A}_w(N+1)]_{n-k}) = ([\boldsymbol{a}_{1,w}]_{n-k}, \dots, [\boldsymbol{a}_{N+1,w}]_{n-k});$$
   $$([\boldsymbol{B}_w(1)]_{n-k}, \dots, [\boldsymbol{B}_w(N+1)]_{n-k}) = ([\boldsymbol{b}_{1,w}]_{n-k}, \dots, [\boldsymbol{b}_{N+1,w}]_{n-k});$$
   $$([\boldsymbol{C}_w(1)]_{n-k}, \dots, [\boldsymbol{C}_w(N+1)]_{n-k}) = ([\boldsymbol{c}_{1,w}]_{n-k}, \dots, [\boldsymbol{c}_{N+1,w}]_{n-k}).$$
   They use the former two, respectively, to locally compute $[\boldsymbol{A}_w(l)]_{n-k}$ and $[\boldsymbol{B}_w(l)]_{n-k}$ for $l \in [N+2, m]$ (using Lagrange interpolation).
3. Then, for $l \in [N+2, m]$, the parties execute $\pi_{\mathsf{beaver}}$ on input $[\boldsymbol{A}_w(l)]_{n-k}$ and $[\boldsymbol{B}_w(l)]_{n-k}$, using packed beaver triples $[\boldsymbol{a}_{l,w}]_{n-k}, [\boldsymbol{b}_{l,w}]_{n-k}, [\boldsymbol{c}_{l,w}]_{n-k}$, to receive $[\boldsymbol{C}_w(l)]_{n-1}$.
4. Note that the sharings $([\boldsymbol{C}_w(1)]_{n-1}, \dots, [\boldsymbol{C}_w(m)]_{n-1})$ implicitly define the polynomials $\boldsymbol{C}_w(\cdot) = \boldsymbol{A}_w(\cdot) * \boldsymbol{B}_w(\cdot)$. Thus, for $l \in [m+1, m+\mu]$, the parties locally compute and output $([\boldsymbol{a}^{l,w}]_{n-k}, [\boldsymbol{b}^{l,w}]_{n-k}, [\boldsymbol{c}^{l,w}]_{n-1}) \leftarrow ([\boldsymbol{A}_w(m+l)]_{n-k}, [\boldsymbol{B}_w(m+l)]_{n-k}, [\boldsymbol{C}_w(m+l)]_{n-1})$ (using Lagrange interpolation).

**Checking the degree of $P_1$'s sharings**: The parties invoke $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ on all sharings sent by $P_1$ in $\pi_{\mathsf{beaver}}$.

*Communication complexity of $\Pi_{\mathsf{triple}}$.* In extraction level 1:

1. For each $l \in [m]$, step 3 costs $\ell^2 \cdot |\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}|$ with all-but-negligible probability, since the L0 row norm of $\boldsymbol{M}$ is $\ell$ with all-but-negligible probability..
2. For each $w \in [\lfloor t/k \rfloor]$, Step 5a costs $2n^2$ total.
3. For each $w \in [\lfloor t/k \rfloor], l \in [m]$, step 5(c)i costs $\ell \cdot n$ total.

In extraction level 2, step 3 runs $\pi_{\mathsf{beaver}} \leq m/2$ times. Each run of $\pi_{\mathsf{beaver}}$ costs $2n$ elements for Step 2 and $2n$ elements for Step 3, for $4n$ total Thus, in total, the runs of $\pi_{\mathsf{beaver}}$ cost $\leq m/2 \cdot 4n$ elements.

The above totals at most $m \cdot \ell^2 \cdot |\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}| + (t/k) \cdot (2n^2 + m \cdot \ell \cdot n) + 2nm$ elements. Using our weakly super-invertible matrix family $\mathcal{M}_{\zeta\mathbf{col}}$ from Section 5.2, we have that $m = O(n)$ and $\ell = O(\lambda)$. Also, $k = \Omega(n)$, so the above cost is $O(n \cdot \lambda^2 \cdot |\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}| + t \cdot n \cdot \lambda)$. Furthermore, if $|\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}| = \widetilde{O}(t)$ (e.g., using [CLOS02]), then this is $\widetilde{O}(n \cdot t)$. This cost is for $\mu \cdot \lfloor t/k \rfloor = \Omega(t)$ packed beaver triples, which is $\widetilde{O}(n)$ per packed triple, or $\widetilde{O}(1)$ per individual triple.

**Lemma 6.** *$\Pi_{\mathsf{triple}}$ UC-realizes $\mathcal{F}_{\mathsf{triple}}$ in the $(\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{verify\text{-}deg}})$-hybrid model, for $2\varepsilon n > m = \Omega(n)$ and any $1/2 < \gamma < 1$.*

See proof of the Lemma in Appendix D.

### 6.4 Authenticated Packed Shamir Sharings

Now we present and instantiate the functionality and procedure that are used to generate authenticated packed Shamir sharings.

**Generating Random Authenticated Packed Shamir Sharings.** First, we introduce $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ which is used to generate authenticated packed Shamir sharings of random values. Let $m \in \mathbb{N}, 1/2 < \gamma < 1$, $\mu \leftarrow \gamma \cdot m - (m+1)/2$, and $t \geq k$. $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ takes in the parties' shares of the MAC key $[\boldsymbol{\gamma}]_{n-k}$, reconstructs $\boldsymbol{\gamma}$, and finally samples $\mu \cdot \lfloor t/k \rfloor$ random authenticated packed sharings $[\![\boldsymbol{r}_{l,w}]\!]_{n-k} := ([\boldsymbol{r}_{l,w}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}]_{n-k})$ consistent with shares input by the adversary.

**Functionality 4: $\mathcal{F}_{\mathsf{auth\text{-}rand}}$**

**Parameters**: Let $m \in \mathbb{N}, 1/2 < \gamma < 1$, $\mu \leftarrow \gamma \cdot m - (m+1)/2$, and $t \geq k$.

1. $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ first receives from all parties their shares of $[\boldsymbol{\gamma}]_{n-k}$ and reconstructs $\boldsymbol{\gamma}$.
2. $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ then receives from the adversary either $\mathsf{abort}$ or the corrupted parties' shares $(r_{l,w}^j, s_{l,w}^j)_{l \in [\mu], w \in [\lfloor t/k \rfloor], j \in \mathsf{Corr}}$.
3. Finally, $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ samples random sharings $([\boldsymbol{r}_{l,w}]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}]_{n-k})_{l \in [\mu], w \in [\lfloor t/k \rfloor]}$ based on the corrupted parties' shares and distributes to the honest parties their shares.

We now present our protocol $\Pi_{\mathsf{auth\text{-}rand}}$ which instantiates $\mathcal{F}_{\mathsf{auth\text{-}rand}}$. $\Pi_{\mathsf{auth\text{-}rand}}$ first invokes $\mathcal{F}_{\mathsf{triple}}$ to get $\mu \cdot \lfloor t/k \rfloor$ random triples $([\boldsymbol{r}]_{n-k}, [\boldsymbol{s}]_{n-k}, [\boldsymbol{r} * \boldsymbol{s} + \boldsymbol{\delta}]_{n-1})$.

For each such triple, the parties first compute $[\boldsymbol{\gamma}+\boldsymbol{s}]_{n-k}$ and open it to $P_1$. $P_1$ then reconstructs $\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}$ and computes then distributes packed Shamir sharing $[\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{k-1}$. Finally, the parties compute $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-1} \leftarrow [\boldsymbol{r}_{l,w}]_{n-k} * [\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{k-1} - [\boldsymbol{r}_{l,w}*\boldsymbol{s}_{l,w}+\boldsymbol{\delta}_{l,w}]_{n-1}$ and invoke $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ on these obtained values to receive back $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-k}$. The parties then output each such $[\![\boldsymbol{r}]\!]_{n-k} \leftarrow ([\boldsymbol{r}]_{n-k}, [\boldsymbol{\gamma}*\boldsymbol{r}-\boldsymbol{\delta}]_{n-k})$.

Note that since $n-k>n-t$, the honest parties' shares do not define any given sharing, and thus the adversary can always locally change the corrupted parties' shares to change the underlying secret. We may equivalently think that the shares of corrupted parties are changed so that the secret of the MAC sharing is $[\boldsymbol{\gamma}*\boldsymbol{r}]_{n-k}$ (and they can then change their shares to any arbitrary values). Thus, the output authenticated sharings satisfy those demanded by the functionality.

**Protocol 3: $\mathit{\Pi}_{\mathsf{auth\text{-}rand}}$**

1. The parties take as input $[\boldsymbol{\gamma}]_{n-k}$.
2. The parties invoke $\mathcal{F}_{\mathsf{triple}}$ and receive random sharings $(([\boldsymbol{r}_{1,w}]_{n-k}, [\boldsymbol{s}_{1,w}]_{n-k}, [\boldsymbol{r}_{1,w} * \boldsymbol{s}_{1,w} + \boldsymbol{\delta}_{1,w}]_{n-1}), \ldots, ([\boldsymbol{r}_{\mu,w}]_{n-k}, [\boldsymbol{s}_{\mu,w}]_{n-k}, [\boldsymbol{r}_{\mu,w} * \boldsymbol{s}_{\mu,w} + \boldsymbol{\delta}_{\mu,w}]_{n-1}))_{w\in[\lfloor t/k\rfloor]}$.
3. For $w \in [\lfloor t/k\rfloor], l \in [\mu]$:
   (a) The parties first compute $[\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{n-k}$ and open it to $P_1$.
   (b) $P_1$ next reconstructs $\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}$ and computes then distributes packed Shamir sharing $[\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{k-1}$.
   (c) Then, the parties compute $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-1} \leftarrow [\boldsymbol{r}_{l,w}]_{n-k} * [\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{k-1} - [\boldsymbol{r}_{l,w}*\boldsymbol{s}_{l,w}+\boldsymbol{\delta}_{l,w}]_{n-1}$ and invoke $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ on it and receive back $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-k}$. [a]
4. Next, the parties invoke $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ on the $[\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{k-1}$ sharings.
5. Finally, the parties output $[\![\boldsymbol{r}_{l,w}]\!]_{n-k} \leftarrow ([\boldsymbol{r}_{l,w}]_{n-k}, [\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-k})$, for $l\in[\mu], w\in[\lfloor t/k\rfloor]$.

[a] Really, the parties input $n-t$ sharings in parallel to $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ at a time.

*Communication complexity of $\mathit{\Pi}_{\mathsf{auth\text{-}rand}}$.* Step 2 costs $\widetilde{O}(n)$ communication and OLE correlations per triple, Step 3a costs $n$ elements per sharing, Step 3b costs $n$ elements per sharing, and Step 3c costs $O(n)$ elements per sharing. The above totals $\widetilde{O}(n)$ communication and OLE correlations per sharing, or $\widetilde{O}(1)$ per individual slot.

**Lemma 7.** *$\mathit{\Pi}_{\mathsf{auth\text{-}rand}}$ UC-realizes $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ in the $(\mathcal{F}_{\mathsf{triple}}, \mathcal{F}_{\mathsf{deg\text{-}reduce}})$-hybrid model.*

See proof of the Lemma in Appendix D.

**Authenticating Packed Shamir Sharings.** Now we present procedure $\pi_{\mathsf{auth}}$ which authenticates input packed sharings. It does so by generating random authenticated packed sharings using $\mathcal{F}_{\mathsf{auth\text{-}rand}}$, and then using the common mask-and-open technique to authenticate the input sharings using the authentication of those from $\mathcal{F}_{\mathsf{auth\text{-}rand}}$.

**Procedure 4: $\pi_{\mathsf{auth}}$**

1. The parties take as input $([\boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{u}_{n-t}]_{n-k})$, and $[\boldsymbol{\gamma}]_{n-k}$.
2. First, the parties invoke $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ on input $[\boldsymbol{\gamma}]_{n-k}$ to prepare random authenticated sharings $([\![\boldsymbol{r}_1]\!]_{n-k}, \ldots, [\![\boldsymbol{r}_{n-t}]\!]_{n-k})$.
3. Then for $l \in [n-t]$:
   (a) They first compute $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-k}$ and open it to $P_1$.
   (b) $P_1$ next reconstructs $\boldsymbol{u}_l + \boldsymbol{r}_l$ and computes then distributes packed Shamir sharing $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$.
   (c) Then, the parties compute $[\boldsymbol{\gamma} * \boldsymbol{u}_l]_{n-1} \leftarrow [\boldsymbol{\gamma}]_{n-k} * [\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1} - [\boldsymbol{\gamma} * \boldsymbol{r}_l]_{n-k}$.
4. Then they invoke $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ on input $([\boldsymbol{\gamma} * \boldsymbol{u}_1]_{n-1}, \ldots, [\boldsymbol{\gamma} * \boldsymbol{u}_{n-t}]_{n-1})$ and receive back $([\boldsymbol{\gamma} * \boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{\gamma} * \boldsymbol{u}_{n-t}]_{n-k})$.
5. Next, they invoke $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ on all of the $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$ sharings.
6. Finally, the parties output $[\![\boldsymbol{u}_l]\!]_{n-k} \leftarrow ([\boldsymbol{u}_l]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{u}_l]_{n-k})$, for $l \in [n-t]$.

*Communication complexity of* $\pi_{\mathsf{auth}}$.

1. Step 2 costs $\widetilde{O}(n)$ communication and OLE correlations per sharing.
2. Step 3a costs $n$ elements per sharing.
3. Step 3b costs $n$ elements per sharing.
4. Step 4 costs $O(n)$ elements per sharing.

The above totals $\widetilde{O}(n)$ communication and OLE correlations per sharing, or $\widetilde{O}(1)$ per individual slot.

## 6.5 The Preprocessing Protocol

Finally, we describe the rest of $\mathit{\Pi}_{\mathsf{prep\text{-}mal}}$. Notably, for packed, authenticated beaver triples, $\mathit{\Pi}_{\mathsf{prep\text{-}mal}}$ first in invokes $\mathcal{F}_{\mathsf{triple}}$ to receive (unauthenticated) packed beaver triples. Then, after degree-reducing the '$\boldsymbol{c}$ components', $\mathit{\Pi}_{\mathsf{prep\text{-}mal}}$ uses $\pi_{\mathsf{auth}}$ to authenticate each component of the triples, followed by running $\pi_{\mathsf{sacrifice}}$ on them, along with the associated checks of $\pi_{\mathsf{check\text{-}zero}}$ and $\mathcal{F}_{\mathsf{verify\text{-}deg}}$, to ensure that none of them have any error.

The parties use a similar process to generate the random authenticated sharings for the input and output gates. Also, the parties use $\mathcal{F}_{\mathsf{rand}}$ to sample some random sharings needed throughout, including for sampling a sharing of the MAC key, $[\boldsymbol{\gamma}]_{n-k}$.

Note that since $n-k > n-t$, the honest parties' shares do not define any given sharing, and thus the adversary can always locally change the corrupted parties' shares to change the underlying secret. We may equivalently think that the shares of corrupted parties are changed so that the secret of the sharings $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-k}$ for input gates is actually $\boldsymbol{\Delta}_l * \boldsymbol{r}_l$ (and they can then change their shares to any arbitrary values). Thus, the output sharings satisfy those demanded by the functionality (see the proof of Theorem 6 below for more).

**Protocol 5: $\Pi_{\mathsf{prep\text{-}mal}}$**

1. **Sampling random MAC key**: All parties invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare a random sharing of the form $[\boldsymbol{\gamma}]_{n-k}$, where $\boldsymbol{\gamma} = (\gamma, \ldots, \gamma)$ (and they ignore all other output sharings from $\mathcal{F}_{\mathsf{rand}}$).
2. **Preparing Random, Packed, Authenticated Sharings for Input and Output Gates**: Let $N_1$ denote the number of input and output gates.
   (a) The parties first invoke $\mathcal{F}_{\mathsf{triple}}$ to prepare $N_1/k$ packed Beaver triples $[\boldsymbol{\Delta}_l]_{n-k}, [\boldsymbol{r}_l]_{n-k}, [\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-1}$, for $l \in [N_1/k]$.
   (b) Then the parties input the $[\boldsymbol{r}_l]_{n-k}$ ($n-t$ at a time) to $\pi_{\mathsf{auth}}$ to obtain authenticated random packed sharings $[\![\boldsymbol{r}_l]\!]_{n-k}$.
   (c) Next, they invoke $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ on input $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-1}$ ($n-t$ at a time) to obtain $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-k}$.
   (d) Additionally, letting $N_o$ be the number of output gates, the parties invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare $N_o/k$ random sharings of the form $[\mathbf{0}]_{n-1}$.
3. **Preparing Packed, Authenticated Beaver Triples**: Let $N_2$ denote the number of multiplication gates. Repeat the following, until $\geq N_2/k$ packed triples are generated:
   (a) All parties invoke $\mathcal{F}_{\mathsf{triple}}$ twice times to generate $2\mu$ packed Beaver triples $[\boldsymbol{a}_l]_{n-k}, [\boldsymbol{b}_l]_{n-k}, [\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-1}$ for $l \in [2\mu]$.
   (b) Next, they invoke $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ on input $[\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-1}$ ($n-t$ at a time, in parallel) to obtain $[\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-k}$.
   (c) The parties input each component of $[\boldsymbol{a}_l]_{n-k}, [\boldsymbol{b}_l]_{n-k}, [\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-k}$, for $l \in [2\mu]$, ($n-t$ at a time) to $\pi_{\mathsf{auth}}$ to obtain $[\![\boldsymbol{a}_l]\!]_{n-k}, [\![\boldsymbol{b}_l]\!]_{n-k}, [\![\boldsymbol{c}_l + \boldsymbol{\delta}_l]\!]_{n-k}$.
   (d) Then, all parties run $\pi_{\mathsf{sacrifice}}$ on $[\![\boldsymbol{a}_l]\!]_{n-k}, [\![\boldsymbol{b}_l]\!]_{n-k}, [\![\boldsymbol{c}_l + \boldsymbol{\delta}_l]\!]_{n-k}$ and $[\![\boldsymbol{a}_{\mu+l}]\!]_{n-k}, [\![\boldsymbol{b}_{\mu+l}]\!]_{n-k}, [\![\boldsymbol{c}_{\mu+l} + \boldsymbol{\delta}_{\mu+l}]\!]_{n-k}$ to compute $[\![\boldsymbol{\gamma} * \boldsymbol{\theta}_l]\!]_{n-1}$, for $l \in [\mu]$, then input these to $\pi_{\mathsf{check\text{-}zero}}$ to verify the correctness of the triples, and finally invoke $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ on input all of the degree-$(k-1)$ sharings from $\pi_{\mathsf{sacrifice}}$.
   (e) If the parties do not abort, this must mean that each $\boldsymbol{\delta}_l = \mathbf{0}$, and the parties output $[\![\boldsymbol{a}_l]\!]_{n-k}, [\![\boldsymbol{b}_l]\!]_{n-k}, [\![\boldsymbol{c}_l]\!]_{n-k}$ for $l \in [\mu]$.
   (f) Additionally, the parties invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare sharings $[\mathbf{0}_1]_{n-1}$ and $[\mathbf{0}_2]_{n-1}$, for each group of $k = \Omega(n)$ multiplication gates.
4. **Preparing Random, Packed Sharings for the Computation Verification**:
   (a) The parties first invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare two random sharings $[\mathbf{0}_1]_{n-1}$ and $[\mathbf{0}_2]_{n-1}$.
   (b) Next, they invoke $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ on input $[\boldsymbol{\gamma}]_{n-k}$ to obtain a random authenticated packed sharing $[\![\boldsymbol{r}]\!]_{n-k}$ (and they ignore the rest).
   (c) Finally, they locally convert $[\boldsymbol{r}]_{n-k}$ and $[\boldsymbol{\gamma} * \boldsymbol{r}]_{n-k}$ into additive sharings $(\langle r \rangle, \langle \gamma \cdot r \rangle)$ of the respective first packed secrets, and use $\mathcal{F}_{\mathsf{rand}}$ to refresh them.[a]

[a] We show how to do this in Appendix C.

*Communication complexity of $\Pi_{\mathsf{prep\text{-}mal}}$.* We will focus on the communication complexity needed to generate the preprocessing required for multiplication gates:

1. Step 3a consumes invokes $\mathcal{F}_{\mathsf{triple}}$, which costs $\widetilde{O}(n)$ communication and OLE correlations per packed triple.
2. Step 3b invokes $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$, which costs $O(n)$ communication and OLE correlations per sharing.
3. Step 3c runs $\pi_{\mathsf{auth}}$, which costs $\widetilde{O}(n)$ communication and OLE correlations per sharing.
4. Step 3d runs $\pi_{\mathsf{sacrifice}}$, which costs $O(n)$ per pair of triples. We ignore the costs of $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ and $\pi_{\mathsf{check\text{-}zero}}$, since they are independent of $\gamma \cdot m$.
5. Step 3f invokes $\mathcal{F}_{\mathsf{rand}}$ for $\Sigma$ such that $|\Sigma| = 1$, which costs $O(n)$ per sharing.

This totals $\widetilde{O}(n)$ communication and OLE correlations for every group of $k$ multiplication gates, which, since the overhead per OLE correlation is $\widetilde{O}(1)$ (e.g., by using [CLOS02]), is $\widetilde{O}(1)$ communication per multiplication gate, since $k = \Omega(n)$.

**Theorem 6.** *$\Pi_{\mathsf{prep\text{-}mal}}$ UC-realizes $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ in the ($\mathcal{F}_{\mathsf{rand}}$, $\mathcal{F}_{\mathsf{triple}}$, $\mathcal{F}_{\mathsf{auth\text{-}rand}}$, $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$, $\mathcal{F}_{\mathsf{verify\text{-}deg}}$, $\mathcal{F}_{\mathsf{coin}}$, $\mathcal{F}_{\mathsf{commit}}$)-hybrid model against a malicious, adaptive adversary who controls $t < (1-\varepsilon)n$ parties.*

See proof of the Theorem in Appendix D.

## References

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011.

BCG$^+$19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

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

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

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

BCS19. Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 274–302. Springer, Heidelberg, August 2019.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

BGJK21. Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. Order-C secure multiparty computation for highly repetitive circuits. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 663–693. Springer, Heidelberg, October 2021.

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

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

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

CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

CP17. Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63(1):428–468, 2017.

CsW19. Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 30–60. Springer, Heidelberg, August 2019.

DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 445–465. Springer, Heidelberg, May / June 2010.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

DM02. Olivier Dubois and Jacques Mandler. The 3-xorsat threshold. *Comptes Rendus Mathematique*, 335(11):963–966, 2002.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, August 2007.

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

EGP$^+$23. Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. SuperPack: Dishonest majority MPC with constant online communication. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2023.

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

GIOZ17. Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 420–446. Springer, Heidelberg, August 2017.

GIP15. Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multiparty computation: From passive to active security via secure SIMD circuits. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 721–741. Springer, Heidelberg, August 2015.

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. ACM Press, May 1987.

Goe. Michel Goemans. Chernoff bounds, and some applications. `https://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf`. Accessed: 2024-02-02.

GPS21. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 275–304, Virtual Event, August 2021. Springer, Heidelberg.

GPS22. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2022.

GSY21. S. Dov Gordon, Daniel Starin, and Arkady Yerukhimovich. The more the merrier: Reducing the cost of large scale MPC. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 694–723. Springer, Heidelberg, October 2021.

HIK07. Danny Harnik, Yuval Ishai, and Eyal Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 284–302. Springer, Heidelberg, August 2007.

HN06. Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006.

IPS09. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

KTZ13. Jonathan Katz, Aishwarya Thiruvengadam, and Hong-Sheng Zhou. Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 14–31. Springer, Heidelberg, February / March 2013.

PS16. Boris Pittel and Gregory B Sorkin. The satisfiability threshold for k-xorsat. *Combinatorics, Probability and Computing*, 25(2):236–268, 2016.

RB89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Heidelberg, August 2022.

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

WYKW21. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

Yao82. Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, page 160–164, USA, 1982. IEEE Computer Society.

# Supplementary Material

## A Additional Preliminaries

**Additive Secret Sharings.** An additive sharing of some value $x$ is a vector $(w_1, \ldots, w_n)$ such that $\sum_{i=1}^{n} w_i = x$. We denote such a sharing $\langle x \rangle$. The $i$-th share $w_i$ is held by $P_i$. Reconstruction is done by summing the shares as above. For a random additive sharing $\langle x \rangle$, any $n-1$ shares are independent of $x$. Additive sharings $\langle x_1 \rangle$ and $\langle x_2 \rangle$ can be added together to get $\langle x_1 + x_2 \rangle$ by having parties simply add their shares of the two sharings together.

**Basic Functionalities.** For our protocol, we will use two basic functionalities that are common in the literature. One is $\mathcal{F}_{\mathsf{commit}}$, which enables a party to commit to some values of their choice towards a set of receivers, without revealing the value. Later, the party can open their committed value with the guarantee that this value is exactly the same one that was committed to earlier. One way to instantiate this is to use a random oracle $\mathsf{H}$ (as in [DKL+13]): a party commits to $m$ by sampling random $r \leftarrow_{\$} \{0,1\}^{\kappa}$ and sending $c \leftarrow \mathsf{H}(m, r) \in \{0,1\}^{\lambda}$; then the party opens $m$ by sending the pair $(m, r)$ to the receivers, who check that $\mathsf{H}(m, r) = c$. The communication complexity is $\lambda$ bits for committing and $|m| + \lambda$ bits for opening, both per receiver. The second functionality is $\mathcal{F}_{\mathsf{coin}}$, which provides parties with a uniformly random value $r \in \mathbb{F}$ when invoked. The standard way to instantiate this is by having the parties (i) compute some random sharing unknown to the adversary, $\langle r \rangle$, for example by independently sharing random values $\langle r_1 \rangle, \ldots, \langle r_n \rangle$, then computing $\langle r \rangle \leftarrow \sum_{i=1}^{n} \langle r_i \rangle$; (ii) commit to their individual shares; and (iii) finally open $\langle r \rangle$. If more coins are needed at the same time, $\langle r \rangle$ can be expanded using a PRG.

**Chernoff's Bound.** Throughout our paper, we will use Chernoff's bound that provides tail bounds for the sum of multiple independent binary variables. We provide the statement of the theorem below (see [Goe] for example).

**Theorem 7.** *Let $X = X_1 + \ldots + X_n$ where $X_i = 1$ with probability $p$ and $X_i = 0$ with probability $1 - p$ and all $X_i$ are independent. Let $\mu = \mathbb{E}[X] = pn$. Then,*

1. $\Pr[X > (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta} \cdot \mu}$ *for any* $\delta > 0$.
2. $\Pr[X < (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2} \cdot \mu}$ *for any* $0 < \delta < 1$.

*In other words,* $\Pr[|X - \mu| > \delta \cdot \mu] \leq e^{-\Theta(\mu)}$ *for any constant* $0 < \delta < 1$.

## B Supplementary Material for Invertible Matrices

### B.1 Extending $\mathcal{M}_{\zeta\mathbf{col}}$ to Fields of Size Two

So far, we have shown that $\mathcal{M}_{\zeta\mathbf{col}}$ emits a weakly super-invertible matrix over finite fields satisfying $|\mathbb{F}| \geq 3$. Next, we show that $\mathcal{M}_{\zeta\mathbf{col}}$ is also a weakly super-invertible matrix in the binary field. Although, we note that we require larger choice of $\zeta = \Omega(\lambda + \log m)$. When $|\mathbb{F}| = O(1)$ and $|F| \geq 3$, we only require that $\zeta = \Theta(\log |\mathbb{F}|) = \Theta(1)$ for our construction $\mathcal{M}_{\zeta\mathbf{col}}$. We leave it as an open problem to obtain the same overhead for $|\mathbb{F}| = 2$. For convenience, we will use $\mathbb{F}_2$ for the binary field going forward.

**Theorem 8.** *Pick any choice of constants $c \geq 1$, $0 < \gamma < 1$ and $\gamma < \eta \leq c$. Let the number of columns be $n = c \cdot m$ and the number of non-zero entries in each column be $\zeta = \Theta(\lambda + \log m)$. The $m \times n$ matrix family $\mathcal{M}_{\zeta\mathbf{col}}$ is $(2^{-\lambda}, \gamma, \eta, O(\lambda + \log m))$-weakly super-invertible over $\mathbb{F}_2$ for sufficiently large $m = \Omega(\lambda \log m)$.*

As a reminder, $\mathcal{M}_{\zeta\mathbf{col}}$ places uniformly random non-zero entries into $\zeta$ random locations per column. For the binary field $\mathbb{F}_2$, this means that $\mathcal{M}_{\zeta\mathbf{col}}$ will place 1 entries into $\zeta$ random locations per column.

To do this, we see that the first two steps of the proof may still be used identically without modification. In particular, the relation between null space vectors and rank remains true for every field. Furthermore, the argument in the second step simply calculates the probability that each row is chosen at least twice. This remains a requirement for every field. The main challenge occurs in the third step of the proof. Our proofs of Lemma 4 and Lemma 5 utilize the fact that the underlying field is sufficiently large. For the binary field $\mathbb{F}_2$, we cannot make this assumption anymore.

Instead, we will relate the setting of $\mathbb{F}_2$ to the random XORSAT problem. The XORSAT problem consists of $m$ variables and $h$ equations over the binary field $\mathbb{F}_2$. Each of the $h$ equations are generated by picking $\zeta$ different variables at random and requiring that the sum of the $\zeta$ variables is equal to a uniformly random $b \in \{0, 1\}$. We say a specific XORSAT instance is satisfiable if there exists some assignment to the $m$ variables such that all $n$ equations are satisfied.

Next, we present a connection between $\mathcal{M}_{\zeta\mathbf{col}}$ and the XORSAT problem. Recall that $\mathcal{M}_{\zeta\mathbf{col}}$ generates $m \times n$ matrices $\boldsymbol{M}$ where each column consists of $\zeta$ non-zero entries (in $\mathbb{F}_2$, these will have values 1 and the remaining entries are 0). Consider any sub-matrix $\boldsymbol{M}^C$ for any subset of $h$ columns $C \subseteq [n]$. We can interpret each of the $m$ rows as variables and each of the $h = \eta \cdot m$ columns as equations. Finally, we will pick a uniformly random bit vector $\mathbf{b} \in \{0, 1\}^h$. Then, we can see that $\boldsymbol{M}^C$ and $\mathbf{b}$ is an instance of the XORSAT problem by trying to find whether a solution $\mathbf{x}$ exists such that $(\boldsymbol{M}^C)^\intercal \mathbf{x} = \mathbf{b}$ where $(\boldsymbol{M}^C)^\intercal$ is the transpose of $\boldsymbol{M}^C$.

We will extend our notion of null space vectors to the XORSAT problem. Consider an instance of the XORSAT problem with $m$ variables and $h$ equations denoted by the $h \times m$ matrix $\boldsymbol{A} \in \{0, 1\}^{h \times m}$ and the solution vector $\mathbf{b} \in \{0, 1\}^h$. A null space vector is a subset of rows of $\boldsymbol{A}$ whose component-wise sum is even. In the field $\mathbb{F}_2$, this means that the component-wise sum will be an entirely zero vector. Finally, we note that we can simply ignore the non-zero coefficients for each null space vector. In the binary field $\mathbb{F}_2$, the only non-zero entry is 1 meaning that all the coefficients must be 1.

Going back to the proof, we will use this connection to re-do step 3 of the proof where we will consider the case of large null space vectors of size $x > \beta \cdot m / \zeta$.

We rely on known results in a variant of the XORSAT problem known as the constrained model introduced in [DM02] where each variable must appear in at least two equations. The unconstrained model of XORSAT omits this restriction. The unconstrained model can be reduced to the constrained model by simply removing variables that appear in at most one equation, which is the typical approach for analyzing the unconstrained XORSAT problem (see [PS16] as an example).

For our choice of $\zeta = O(\lambda + \log m)$, we show that the probability that there even exists a single variable that appears in at most one equation is exponentially small in $\lambda$. In other words, $\mathcal{M}_{\zeta\mathbf{col}}$ will produce an instance of XORSAT in the constrained model except with negligible probability.

**Lemma 8.** *Let $\boldsymbol{M}$ be randomly generated by $\mathcal{M}_{\zeta\mathbf{col}}$ with $\zeta = \Theta(\lambda + \log m)$ over the binary field $\mathbb{F}_2$. Every row of $\boldsymbol{M}$ has at least two non-zero entries except with probability $2^{-4\lambda}$.*

*Proof.* Consider any of the $m$ rows and we will apply Chernoff's bound. Each of the $n$ columns will choose to place a 1-entry in the fixed row with independent probability $\zeta/m$. So, the expected number of 1-entries in any row is $n \cdot \zeta/m = \Theta(\zeta)$ that is sum of $n$ independent binary variables. By Chernoff's bounds, we know that the probability that any row has at most a single 1-entry is at most $2^{-2\lambda - \log m}$ for sufficiently large $\zeta = O(\lambda + \log m)$. Applying a Union bound over the $m$ rows obtains the desired probability of $m \cdot 2^{-4\lambda - \log m} \leq 2^{-4\lambda}$. □

As a remark, the reason for the larger choice of $\zeta$ is the ability to guarantee that the matrix corresponds to a XORSAT instance in the constrained model. Next, we are ready to apply known results about XORSAT from prior works. In particular, we will use a result of Pittel and Sorkin [PS16] that proves an upper bound on the expected value of the number of null space vectors of size $x = \Omega(m/\zeta)$:

**Lemma 9 (Lemma 9, [PS16]).** *Suppose $\zeta \geq 3$ and pick any $2/\zeta < \alpha < 1$ chosen independently of $m$ such that $h = \alpha \cdot m$. Let $\boldsymbol{A}$ be an $h \times m$ matrix denoting a random XORSAT instance in the constrained model where each variable appears in at least two equations. For any $x = \Omega(m/\zeta)$, let $Y^x$ denote the number of null space vectors in $\boldsymbol{A}$ of size $x$. Then, $\mathbb{E}[Y^x] \leq 2^{-\Omega(\zeta \cdot x)}$.*

The above lemma considers a single random XORSAT instance $\boldsymbol{A}$ of $h = \alpha \cdot m$ equations and $m$ variables such that the number of equations is a constant fraction smaller than the number of variables. Then, the expected number of null space vectors of size $x$ in $\boldsymbol{A}$ is exponentially small in $\zeta \cdot x$. We show that this is sufficient to show that any matrix generated by our matrix family $\mathcal{M}_{\zeta\mathbf{col}}$ will also not contain null space vectors of size $x$ for any $x = \Omega(m/\zeta)$.

**Lemma 10.** *Pick any constant $c \geq 1$. Let $n = c \cdot m$ and $\zeta = \Theta(\lambda + \log m)$. Let $\boldsymbol{M}$ be a $m \times n$ matrix generated by $\mathcal{M}_{\zeta\mathbf{col}}$ in the binary field $\mathbb{F}_2$. Then, for every constant $0 < \eta < 1$ and $0 < \beta < 1$, the probability there exists at least one null space vector of any size $\beta \cdot m/\zeta \leq x \leq \eta \cdot m$ in $\boldsymbol{M}$ is at most $2^{-2\lambda}$.*

*Proof.* Consider any fixed size $\beta \cdot m/\zeta \leq x \leq \eta \cdot m$ for null space vectors. Let $E$ be the event that the randomly generated matrix $\boldsymbol{M}^\intercal$ satisfies the constrained XORSAT model meaning that each column of $\boldsymbol{M}^\intercal$ has at least two non-zero entries. For now, we will condition on the event $E$ holding. Now, fix any subset of $h = \max\{x, 2m/\zeta\}$ rows denoted by $C \subseteq [n]$ and consider the sub-matrix $\boldsymbol{A} = (\boldsymbol{M}^{\boldsymbol{C}})^\intercal$. Denote $Y_A^x$ as the number of null space vectors of size $x$ in this sub-matrix $\boldsymbol{A}$. By Lemma 9, we know that $\mathbb{E}[Y_A^x \mid E] \leq 2^{-\Omega(\zeta \cdot x)}$. By Markov's inequality, we get that $\Pr[Y_A^x \geq 1 \mid E] \leq 2^{-\Omega(\zeta \cdot x)}$ since $Y_A^x$ is always non-negative. Next, we can apply a Union bound over all $\binom{n}{h} = \binom{cm}{h}$ choices of rows to obtain a bound on the number of null space vectors in $\boldsymbol{M}$ denoted by $Y^x$:

$$\Pr[Y^x \geq 1 \mid E] \leq \binom{cm}{h} \cdot 2^{-\Omega(\zeta \cdot x)} \leq \binom{cm}{O(x)} \cdot 2^{-\Omega(\zeta \cdot x)} \leq 2^{O(x \cdot \log \zeta)} \cdot 2^{-\Omega(\zeta \cdot x)}$$

where we note that $h = O(x \cdot \zeta^{1/\zeta}) = O(x \cdot 2^{\log \zeta/\zeta}) = O(x)$ and use that $x \geq \beta \cdot m/\zeta = \Omega(m/\zeta)$. We also use Stirling's approximation of binomial coefficients such that $\binom{a}{b} \leq (ea/b)^b$. For sufficiently large $\zeta = O(\lambda + \log m)$, we see that $\Pr[Y^x \geq 1 \mid E] \leq 2^{-4\lambda - \log m}$. Finally, we apply a Union bound over all possible choices of $x$ and remove the conditioning on the event $E$ to get that

$$\sum_{x = \beta \cdot m/\zeta}^{\eta \cdot m} \Pr[Y^x \geq 1 \mid E] + \Pr[\neg E] \leq m \cdot 2^{-4\lambda - \log m} + 2^{-4\lambda} = 2^{-2\lambda}$$

where we used Lemma 8 to get that $\Pr[\neg E] \leq 2^{-4\lambda}$. □

The above (combined with the second step of the proof) shows that, even in the binary field $\mathbb{F}_2$, a random matrix generated by $\mathcal{M}_{\zeta\mathbf{col}}$ will not contain null space vectors larger than some constant $c_x$. With this fact, the final step of the proof proceeds identically.

*Proof of Theorem 8.* Follows identically to the proof of Theorem 4 where we replace the usage of Lemma 5 with the above Lemma 10 to rule out the existence of large null space vectors. □

## B.2 Hyper-Invertible Matrices

We present the definition of hyper-invertible matrices here that strengthens super-invertible matrices.

**Definition 5 (Hyper-Invertible Matrices [BTH08]).** *Let $\boldsymbol{M}$ be a matrix of dimension $m \times n$ with $n \geq m$. $\boldsymbol{M}$ is a hyper-invertible matrix if, for every choice of $1 \leq z \leq m$ and for any two subsets $R = \{r_1, \ldots, r_z\} \subseteq [m]$ and $C = \{c_1, \ldots, c_z\} \subseteq [n]$ both of size exactly $z$, the $z \times z$ sub-matrix defined as*

$$\boldsymbol{M}_R^C = \begin{bmatrix} \boldsymbol{M}_{r_1}[c_1] & \boldsymbol{M}_{r_1}[c_2] & \ldots & \boldsymbol{M}_{r_1}[c_z] \\ \boldsymbol{M}_{r_2}[c_1] & \boldsymbol{M}_{r_2}[c_2] & \ldots & \boldsymbol{M}_{r_2}[c_z] \\ \ldots & \ldots & \ldots & \ldots \\ \boldsymbol{M}_{r_z}[c_1] & \boldsymbol{M}_{r_m}[c_2] & \ldots & \boldsymbol{M}_{r_m}[c_z] \end{bmatrix}$$

*consisting of all entries in $\boldsymbol{M}$ that appears in a row specified by $R$ and a column defined by the subset $C$ must be invertible.*

In our definition, we choose to only focus on the setting with at least as many columns as rows, $n \geq m$. One could also consider a less restrictive definition of hyper-invertible matrices where there are no requirements on the dimensions of the matrix. In that case, one could define hyper-invertibility as any $m \times n$ matrix $\boldsymbol{M}$ such that any $z \times z$ sub-matrix is invertible where $1 \leq z \leq \min(m, n)$. We do not consider such matrices as the level of generality is unnecessary for MPC.

### B.3 Impossibility for Weakly Hyper-Invertible Matrices

Previously, we defined the notion of weakly super-invertible matrices that extended the notion of super-invertibility. A natural question may be to also consider the same weakening for hyper-invertible matrices. We will show that there is no advantage for considering this weakening by proving a lower bound on the $L_0$ norm required for weakly hyper-invertible matrices that are also required by hyper-invertible matrices.

We can consider a matrix family $\mathcal{M}$ that generates a random $m \times n$ matrix $\boldsymbol{M}$. Then, for every sufficiently large $z \geq 1/\gamma$, every $z \times z$ sub-matrix of $\boldsymbol{M}$ should have rank $\gamma \cdot z$ for some constant $\gamma > 0$ except with negligible probability over the random choice of $\boldsymbol{M}$. This is same generalization for weakening hyper-invertible matrices.

**Definition 6 (Weakly Hyper-Invertible Matrix Family).** *Fix constants $0 < \delta, \gamma < 1$. Let $\mathcal{M} = (\mathcal{S}, \mathsf{Gen})$ be a matrix family over $m \times n$ matrices where $n \geq m$. $\mathcal{M}$ is $(\delta, \gamma)$-weakly hyper-invertible if, for every $1/\gamma \leq z \leq m$ and every choice of $z$ rows $R \subseteq [m]$ and $z$ columns $C \subseteq [n]$, the following holds:*

$$\Pr_{\mathbf{R}}[\boldsymbol{M}_R^C \text{ has rank at least } \gamma \cdot z \mid \boldsymbol{M} \leftarrow \mathsf{Gen}(\mathbf{R})] \geq 1 - \delta$$

*where $\mathbf{R}$ is a uniformly random bit string.*

Note that we consider $z \geq 1/\gamma$ to ensure that the requirement forces the $z \times z$ sub-matrix to have rank at least $\gamma \cdot z \geq 1$. One could also consider more stringent requirements where $z \times z$ sub-matrices for $z < 1/\gamma$ still require rank at least one. As we are proving lower bounds, weaker restrictions imply stronger results.

We present the following lower bound on the number of $\Omega(n)$ non-zero entries per row required in a $(\delta, \gamma)$-weakly hyper-invertible matrix. In particular, we prove that, if $\delta$ and $\gamma$ are constants, then the expected $L_0$ norm of any row in a weakly hyper-invertible matrix family must be $\Omega(n)$.

**Theorem 9.** *Suppose that matrix family $\mathcal{M} = (\mathcal{S}, \mathsf{Gen})$ is $(\delta, \gamma)$-weakly hyper-invertible for $\delta \leq 2/3$ and $\gamma = \Theta(1)$. Then, the expected $L_0$ norm for every row $i \in [m]$ satisfies $\mathbb{E}_{\mathbf{R}}[L_0(\boldsymbol{M}_i) \mid \boldsymbol{M} \leftarrow \mathsf{Gen}(\mathbf{R})] = \Omega(n)$.*

*Proof.* Consider any matrix $\boldsymbol{M} \in \mathcal{S}$ that satisfies the requirements for weak hyper-invertibility. That is, every $z \times z$ sub-matrix of $\boldsymbol{M}$ has rank at least $\gamma \cdot z$. Pick the smallest integer $t \geq 1/\gamma$. We can partition any $m \times n$ matrix into disjoint $t \times t$ matrices. Thus, we can obtain $\lfloor m/t \rfloor \cdot \lfloor n/t \rfloor = \Omega(\gamma^2 \cdot m \cdot n) = \Omega(m \cdot n)$ disjoint $t \times t$ sub-matrices using $\gamma = \Theta(1)$. By the weakly hyper-invertibility requirement, each $t \times t$ matrix must have rank at least $\gamma \cdot t \geq 1$. Therefore, each disjoint $t \times t$ sub-matrix must contain at least one non-zero entry. Altogether, $\boldsymbol{M}$ must have $L_0$ norm at least $\Omega(m \cdot n)$. Finally, consider the family $\mathcal{M}$ that must produce matrices satisfying the hyper-invertibility requirement with probability at least $1 - \delta$. Therefore, the expected $L_0$ norm of randomly generated matrices by $\mathcal{M}$ must be at least $\Omega((1 - \delta) \cdot m \cdot n) = \Omega(m \cdot n)$ since $\delta \leq 2/3$. Note each of the $m$ rows can have at most $n$ non-zero entries. Therefore, each row must have expectation of at least $\Omega(n)$ non-zero entries. □

From the above, we can immediately observe that Vandermonde matrices are optimal weakly hyper-invertible matrices with respect to $L_0$ norm. In other words, there is no real advantage for considering weaker notions of hyper-invertibility.

Our above lower bound immediately implies the same norm lower bound for hyper-invertible matrices. In other words, every hyper-invertible matrix must have $\Omega(m \cdot n)$ non-zero entries and the Vandermonde matrix is an optimal hyper-invertible matrix with respect to the number of non-zero entries.

## C Supplementary Material for $\Pi_{\textsf{prep-mal}}$

Here we present the components of $\Pi_{\textsf{prep-mal}}$ which we were unable to fit in the main body.

### C.1 Useful Sub-Functionalities and Sub-Procedures

We first describe some important Sub-Functionalities and Sub-Procedures which we make use of in $\Pi_{\textsf{prep-mal}}$.

**Verifying Degrees of Sharings.** The first such functionality that we instantiate is $\mathcal{F}_{\textsf{verify-deg}}$. This functionality takes in the honest parties' shares of $M$ packed Shamir sharings of unknown degrees, $[\boldsymbol{x}_1]_{d_1}, \ldots, [\boldsymbol{x}_M]d_M$. It checks that all of their degrees are $\leq k - 1$ and if not, sends abort to the honest parties.

**Functionality 5: $\mathcal{F}_{\textsf{verify-deg}}$**

1. $\mathcal{F}_{\textsf{verify-deg}}$ receives from the honest parties their shares of $[\boldsymbol{x}_1]_{d_1}, \ldots, [\boldsymbol{x}_M]_{d_M}$.
2. $\mathcal{F}_{\textsf{verify-deg}}$ then reconstructs the sharings and sends the whole sharings $[\boldsymbol{x}_1]_{d_1}, \ldots, [\boldsymbol{x}_M]_{d_M}$ to the adversary.
3. If $\exists l \in [M]$ such that $d_l > k - 1$ or $\mathcal{F}_{\textsf{verify-deg}}$ receives from the adversary abort, $\mathcal{F}_{\textsf{verify-deg}}$ sends abort to the honest parties.

We provide the following standard protocol $\Pi_{\textsf{verify-deg}}$ to instantiate $\mathcal{F}_{\textsf{verify-deg}}$. In $\Pi_{\textsf{verify-deg}}$, the parties simply use $\mathcal{F}_{\textsf{coin}}$ to sample several random values, then take a random linear combination of the $[\boldsymbol{x}_l]_{d_l}$ defined by the sampled values, and open the resulting sharing, $[\boldsymbol{x}]_d$, to each other. Since $k < n - t$, the number of honest parties and thus honest shares, the honest parties can verify that $[\boldsymbol{x}]_d$ is of degree-$k-1$, even if the corrupted parties sends incorrect shares. Indeed, since the random values are sampled after the sharings are formed, with all-but-negligible probability, if any of the original sharings are not degree-$(k-1)$, then nor will $[\boldsymbol{x}]_d$.

**Protocol 6: $\Pi_{\mathsf{verify\text{-}deg}}$**

1. All parties take as input $M$ packed Shamir sharings distributed by $P_1$, denoted by $[\boldsymbol{x}_1]_{d_1}, \ldots, [\boldsymbol{x}_M]_{d_m}$.
2. All parties invoke $\mathcal{F}_{\mathsf{coin}}$ and receive random values $\chi_1, \ldots, \chi_M$.
3. All parties locally compute $[\boldsymbol{x}]_d \leftarrow \sum_{l=1}^{M} \chi_l \cdot [\boldsymbol{x}_l]_{d_l}$, where $d = \max_l\{d_l\}$.
4. All parties open their shares of $[\boldsymbol{x}]_d$ to one another and check that the shares of $[\boldsymbol{x}]_d$ lie on a degree-$(k-1)$ polynomial (i.e., $d = k-1$). If not, all parties abort.

The communication complexity of $\Pi_{\mathsf{verify\text{-}deg}}$ is $O(n^2)$. If $M = \Omega(n)$, then the communication complexity per underlying shared value is $O(1)$.

**Lemma 11.** *$\Pi_{\mathsf{verify\text{-}deg}}$ UC-realizes $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ in the $\mathcal{F}_{\mathsf{coin}}$-hybrid model.*

*Proof.* We begin by defining the simulator $\mathcal{S}$.

1. $\mathcal{S}$ first receives from $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ the whole sharings $[\boldsymbol{x}_1]_{d_1}, \ldots, [\boldsymbol{x}_M]_{d_M}$.
2. $\mathcal{S}$ then emulates $\mathcal{F}_{\mathsf{coin}}$ by sampling random $\chi_1, \ldots, \chi_M$ and sending them to the adversary.
3. Then, $\mathcal{S}$ computes the honest parties' shares of $[\boldsymbol{x}]_{d_{\max}} \leftarrow \sum_{l=1^M} \chi_l [\boldsymbol{x}]_{d_l}$, where $d_{\max} \leftarrow \max_l d_l$.
4. Finally, $\mathcal{S}$ sends to the adversary the honest parties' shares of $[\boldsymbol{x}]_{d_{\max}}$. Then $\mathcal{S}$ receives from the adversary its shares, and if all of the shares (including the honest parties' shares) do not lie on a polynomial of degree at most $k-1$, then $\mathcal{S}$ sends $\mathcal{F}_{\mathsf{verify\text{-}deg}}$, $\mathsf{abort}$.

Now we argue that the real and ideal worlds are indistinguishable. It is clear that from the honest parties' shares, $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ will always be able to reconstruct $[\boldsymbol{x}_1]_{d_1}, \ldots, [\boldsymbol{x}_M]_{d_M}$ and send them to $\mathcal{S}$, which uses the honest parties' shares. Therefore, it is clear that the real and ideal worlds are identical, up until the point of aborting or not.

Now, we will prove that with all-but-negligible probability, an honest party aborts in the ideal world if and only if they do in the real world. $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ aborts if $\exists l^* \in [M]$ such that $d_{l^*} > k-1$. In this case, we will show that the honest parties will also abort in the real world in $\Pi_{\mathsf{verify\text{-}deg}}$. First, note that $n - t > k - 1$ and assume w.l.o.g., that the $(k+1)$-st honest party's share of $[\boldsymbol{x}_{l^*}]_{k-1}$ is nonzero (if all of the honest parties shares are zero, then they define the zero polynomial of degree $0 \leq k-1$). Consider the first $k$ honest parties' shares of $[\boldsymbol{x}]_{k-1}$ computed in $\Pi_{\mathsf{verify\text{-}deg}}$, $x^{i_1}, \ldots, x^{i_k}$. In order for the shares of $[\boldsymbol{x}]_{k-1}$ to lie on a polynomial of degree $k-1$, it must be that the $(k+1)$-st honest party's share, $x^{i_{k+1}} = \sum_{j=1}^{k} \lambda_j \cdot x^{i_j}$ (where the $\lambda_j$ are Lagrange coefficients). Letting $x_1^{i_{k+1}}, \ldots, x_M^{i_{k+1}}$ be the $(k+1)$-st honest party's shares of $[\boldsymbol{x}_1]_{k-1}, \ldots, [\boldsymbol{x}_M]_{k-1}$, respectively, we have that $x^{i_{k+1}} = \sum_{l=1}^{M} \chi_l \cdot x_l^{i_{k+1}}$. Thus, it must be that $\sum_{l=1}^{M} \chi_l \cdot x_l^{i_{k+1}} = \sum_{j=1}^{k} \lambda_j \cdot x^{i_j}$, which in turn means that $\chi_{l^*} = (\sum_{j=1}^{k} \lambda_j \cdot x^{i_j} - \sum_{l \in [M] \setminus \{l^*\}} \chi_l \cdot x_l^{i_{k+1}})/x_{l^*}^{i_{k+1}}$. However, since $\chi_{l^*}$ is chosen independently of the honest parties' shares, this only happens with negligible probability, and so with all-but-negligible probability, the honest parties abort in the real world. $\mathcal{S}$ might also send $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ if in $\Pi_{\mathsf{verify\text{-}deg}}$, the adversary does not send to $\mathcal{S}$ shares that lie on the same degree-$(k-1)$ polynomial as the honest parties' shares. However, since the honest parties' shares are identical to what they would be in the real world, the honest parties would also abort in this case in the real world. It is also clear that if honest parties do not abort in the ideal world, they also do not abort in the real world. $\square$

**Generating Random Sharings.** The next functionality is $\mathcal{F}_{\mathsf{rand}}$. For any linear secret sharing scheme in $\mathbb{F}$, $\Sigma$, $\mathcal{F}_{\mathsf{rand}}$ samples $n-t$ random $\Sigma$-sharings, each of which are consistent with shares for the corrupted parties that are input by the adversary.

**Functionality 6: $\mathcal{F}_{\mathsf{rand}}(\Sigma)$**

1. $\mathcal{F}_{\mathsf{rand}}$ receives from the adversary the corrupted parties' shares, $\{(r_j^{(1)}, \ldots, r_j^{(n-t)})\}_{j \in \mathsf{Corr}}$.
2. $\mathcal{F}_{\mathsf{rand}}$ samples random $r^{(1)}, \ldots, r^{(n-t)}$.
3. $\mathcal{F}_{\mathsf{rand}}$ samples random sharings $(\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})$ with secrets $r^{(1)}, \ldots, r^{(n-t)}$ such that for all $j \in \mathsf{Corr}$, the $j$-th shares are $(r_j^{(1)}, \ldots, r_j^{(n-t)})$.
4. $\mathcal{F}_{\mathsf{rand}}$ then distributes to the honest parties their shares.

We provide the following standard protocol $\Pi_{\mathsf{rand}}$ to instantiate $\mathcal{F}_{\mathsf{rand}}$. In $\Pi_{\mathsf{rand}}$, each party $P_i$ first samples their own random $\Sigma$-sharing $\boldsymbol{S}^{(i)}$ and distributed their shares to other parties. Then, using some agreed upon $(n-t) \times n$ super-invertible matrix $\boldsymbol{M}$ (for example, the transpose of a Vandermonde matrix), the parties multiply the sharings by $\boldsymbol{M}$ to obtain $n-t$ sharings $\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)}$, which they output. Intuitively, there are at least $n-t$ honest parties, and $\boldsymbol{M}$ is super-invertible, so the randomness of the $n-t$ output sharings has a one-to-one relationship with the randomness of any $n-t$ honest parties' original sharings.

**Protocol 7: $\Pi_{\mathsf{rand}}(\Sigma)$**

1. All parties agree on $(n-t) \times n$ super-invertible matrix $\boldsymbol{M}$.
2. Each party $P_i$ samples a random $\Sigma$-sharing $\boldsymbol{S}^{(i)}$ and distributes the shares to other parties.
3. All parties locally compute $(\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})^{\intercal} \leftarrow \boldsymbol{M} \cdot (\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(n)})^{\intercal}$ and output $(\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})$.

*Communication complexity of $\Pi_{\mathsf{rand}}$.* We let $|\Sigma|$ be the size of the individual shares of $\Sigma$. Then, $\Pi_{\mathsf{rand}}$ costs $n^2 \cdot |\Sigma|$, or $O(n \cdot |\Sigma|)$ per output sharing, since $n - t > \varepsilon \cdot n = \Theta(n)$. This is $O(|\Sigma|)$ per individual shared random value in the sharings.

**Lemma 12.** *$\Pi_{\mathsf{rand}}$ UC-realizes $\mathcal{F}_{\mathsf{rand}}$.*

*Proof.* First recall that we only use sharings which satisfy that there exists some $d \geq t$ such that any $t$ shares are independent of any other set of $d - t$ shares. Now we define the simulator $\mathcal{S}$:

1. First, for each honest party $P_i$, $\mathcal{S}$ samples random sharing $\boldsymbol{S}^{(i)}$, then sends the corrupted parties their shares of the sharings.
2. $\mathcal{S}$ then receives from the adversary the honest parties' shares of the corrupted parties' sharings $\boldsymbol{S}^{(j)}$. $\mathcal{S}$ then for $j \in \mathsf{Corr}$ samples a random $\Sigma$-sharing based on the honest parties shares and views it as the sharing generated by $P_j$.
3. $\mathcal{S}$ then computes $(\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})^{\intercal} \leftarrow \boldsymbol{M} \cdot (\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(n)})^{\intercal}$ for the corrupted parties, and sends their shares to $\mathcal{F}_{\mathsf{rand}}$.

Next we argue that the real and ideal worlds are distributed identically. It is clear that the honest parties distributing their random $\Sigma$-sharings $\boldsymbol{S}^{(i)}$ is identical in both worlds. All that remains is to argue that the shares that the honest parties output in the ideal world are identically distributed to that of the real world. For this we will use the power of super-invertible matrices.

Let $t'$ be the number of corrupted parties after the sharing step, $\{i_1, \ldots, i_{n-t}\} = H \subseteq \mathsf{Hon}$ be any subset of $\mathsf{Hon}$ of size $|H| = n - t$ and $\{j_1, \ldots, j_t\} = \overline{H} = [n] \setminus H$. Let $\boldsymbol{M}^H$ be the $(n-t) \times (n-t)$ sub-matrix of $\boldsymbol{M}$ whose columns correspond to honest parties' indices in $H$. We know that $\boldsymbol{M}^H$ is invertible. Thus, we can write

$$(\boldsymbol{S}^{(i_1)}, \ldots, \boldsymbol{S}^{(i_{n-t})})^{\intercal} = (\boldsymbol{M}^H)^{-1} \cdot ((\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})^{\intercal} - \boldsymbol{M}^{\overline{H}} \cdot (\boldsymbol{S}^{(j_1)}, \ldots, \boldsymbol{S}^{(j_t)})^{\intercal}).$$

This means that the distribution of $(\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})$ is equivalent to that of $(\boldsymbol{S}^{(i_1)}, \ldots, \boldsymbol{S}^{(i_{n-t})})$, given any fixed $\boldsymbol{S}^{(j_1)}, \ldots, \boldsymbol{S}^{(j_t)}$. So, for any fixed sharings $\boldsymbol{S}^{(j_1)}, \ldots, \boldsymbol{S}^{(j_t)}$, the distribution of $(\boldsymbol{R}^{(1)}, \ldots, \boldsymbol{R}^{(n-t)})$ is random given the shares of corrupted parties. This means that in the real world, the shares that honest parties output are random given those of the corrupted parties. In the ideal world, $\mathcal{S}$ generates the shares of each $\boldsymbol{R}^{(i)}$ of the corrupted parties by randomly sampling their shares of the $\boldsymbol{S}^{(j_1)}, \ldots, \boldsymbol{S}^{(j_t)}$ based on the shares of honest parties. Then, the shares of each $\boldsymbol{R}^{(i)}$ of the honest parties are randomly sampled based on the shares of the corrupted parties. Thus, the distribution of the shares of the honest parties in both worlds is identical. $\square$

**Transforming Packed Shamir Sharings to Additive Sharings.** Note that if the parties hold some packed Shamir secret sharing $[\boldsymbol{r}]_{n-k} = (w_1, \ldots, w_n)$, then they can locally convert it to an additive sharing $\langle r_i \rangle = (v_1, \ldots, v_n)$ of the $i$-th element of $\boldsymbol{r}$. Indeed, letting $L_j(-i+1)$ be the $j$-th Lagrange coefficient, party $P_j$ sets their share of $\langle r_i \rangle$ to be $v_j \leftarrow L_j(-i+1) \cdot w_j$. This works since

$\sum_{j=1}^{n} L_j(-i+1)\cdot w_j = r_i$, by definition. However, the sharing $\langle r_i\rangle$ is not uniformly random.

To get a uniformly random sharing, the parties can invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare a random sharing of the form $\langle 0\rangle$, then reset $\langle r_i\rangle \leftarrow \langle r_i\rangle + \langle 0\rangle$.

**Degree Reduction.** Functionality $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ takes in $n-t$ degree-$(n-1)$ packed Shamir sharings $[\boldsymbol{u}_l]_{n-1}$ from the parties and samples $n-t$ degree-$(n-k)$ random packed Shamir sharings of the same underlying secrets, each of which are consistent with corrupted parties' shares that are input by the adversary.

**Functionality 7: $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$**

1. $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ first receives from all parties their shares of the sharings $\{[\boldsymbol{u}_1]_{n-1}, \ldots, [\boldsymbol{u}_{n-t}]_{n-1}\}$ and reconstructs $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-t}$.
2. $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ then receives from the adversary the corrupted parties' shares $\{u_1^j, \ldots, u_{n-t}^j\}_{j\in\mathsf{Corr}}$ of the new sharings.
3. Finally, $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ samples random sharings $[\boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{u}_{n-t}]_{n-k}$ based on the corrupted parties' shares and distributes to the honest parties their shares.

To instantiate $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$, we use the following standard technique. First, the parties invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare $n-t$ random sharings of the form $([\boldsymbol{r}_l]_{n-k}, [\boldsymbol{r}_l]_{n-1})$, for $l \in [n-t]$. Then, the parties for each $l$ compute $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-1}$ and open the sharings to $P_1$. Since the sharing $[\boldsymbol{r}_l]_{n-1}$ is random, so too is $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-1}$, thus nothing is revealed about $\boldsymbol{u}_l$. Then $P_1$ reconstructs each $\boldsymbol{u}_l + \boldsymbol{r}_l$ and distributes sharings $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$ to the other parties, from which they can compute $[\boldsymbol{u}_l]_{n-k}$ by subtracting $[\boldsymbol{r}]_{n-k}$. We use $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ to ensure that $P_1$ did not cheat by sending a sharing of a higher degree. We present the protocol $\Pi_{\mathsf{deg\text{-}reduce}}$ below.

**Protocol 8: $\Pi_{\mathsf{deg\text{-}reduce}}$**

1. All parties hold degree-$(n-1)$ packed Shamir sharings $[\boldsymbol{u}_1]_{n-1}, \ldots, [\boldsymbol{u}_{n-t}]_{n-1}$.
2. They first invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare random sharings $([\boldsymbol{r}_1]_{n-k}, [\boldsymbol{r}_1]_{n-1}), \ldots, ([\boldsymbol{r}_{n-t}]_{n-k}, [\boldsymbol{r}_{n-t}]_{n-1})$.
3. For all $l \in [n-t]$, all parties locally compute $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-1} \leftarrow [\boldsymbol{u}_l]_{n-1} + [\boldsymbol{r}_l]_{n-1}$ and open it to $P_1$.
4. $P_1$ locally reconstructs each $\boldsymbol{u}_l + \boldsymbol{r}_l$ and then distributes sharings $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$.
5. Next, the parties invoke $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ on the $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$ sharings.
6. Finally, all parties locally compute for each $l \in [n-t]$, $[\boldsymbol{u}_l]_{n-k} \leftarrow [\boldsymbol{u}_l + \boldsymbol{r}_l]_{2k-2} - [\boldsymbol{r}_l]_{n-k}$.

*Communication complexity of $\Pi_{\mathsf{deg\text{-}reduce}}$.* We have that $|\Sigma| = 2$, for the random double sharings prepared by $\mathcal{F}_{\mathsf{rand}}$. Thus, using the communication complexity of our $\Pi_{\mathsf{rand}}$, we have that

1. Step 2 costs $2n^2$ elements.
2. Step 3 costs $n \cdot (n-t)$ elements.

3. Step 4 costs $n \cdot (n-t)$ elements.

The above totals $2n^2 + 2n(n-t)$, or $O(n)$ per sharing, since $n - t = \Theta(n)$. This is $O(1)$ per underlying slot.

**Lemma 13.** *$\Pi_{\mathsf{deg\text{-}reduce}}$ UC-realizes $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ in the $\mathcal{F}_{\mathsf{rand}}$-hybrid model.*

*Proof.* First we define the simulator $\mathcal{S}$:

1. $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary $\{(s_j^1, r_j^1), \ldots, (s_j^{n-t}, r_j^{n-t})\}$ and sampling random sharings $([\boldsymbol{r}_1]_{n-k}, [\boldsymbol{r}_1]_{n-1}), \ldots, ([\boldsymbol{r}_{n-t}]_{n-k}, [\boldsymbol{r}_{n-t}]_{n-1})$ consistent with the above as the corrupt parties' shares. Note that if the adversary corrupts additional parties after the invocation of $\mathcal{F}_{\mathsf{rand}}$, $\mathcal{S}$ can just send them their shares.
2. Next:
   (a) If $P_1$ is corrupt, for $l \in [n-t]$, $\mathcal{S}$ sends the adversary random shares $t_l^i$ on behalf of each honest party.
   (b) If $P_1$ is honest, $\mathcal{S}$ receives from the adversary the corrupted parties' shares of $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-1}$. Using the corrupt parties' shares of $[\boldsymbol{r}_l]_{n-1}$ received from the adversary above, $\mathcal{S}$ computes their shares of each $[\boldsymbol{u}_l]_{n-1}$, then sends them to $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$.
3. Then:
   (a) If $P_1$ is corrupt, then $\mathcal{S}$ receives from the adversary the honest parties' shares of each $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$. $\mathcal{S}$ can then reconstruct $\boldsymbol{u}_l + \boldsymbol{r}_l$ using the first $k$ honest parties' shares and since it sampled $\boldsymbol{r}_l$ above, compute $\boldsymbol{u}_l$. Then, $\mathcal{S}$ can compute for each honest party $u_l^i \leftarrow t_l^i - r_l^i$ and based on these and $\boldsymbol{u}_l$, samples the corrupt parties' shares of each $[\boldsymbol{u}_l]_{n-1}$ and sends them to $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$.
   (b) If $P_1$ is honest, $\mathcal{S}$ samples random sharings $[\boldsymbol{t}_l]_{k-1}$ and sends the corrupted parties their shares.
4. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ by first sending the adversary the corrupted parties' shares of the sharings $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$ then aborting if $P_1$ is corrupted and sent sharings of degree greater than $k-1$.
5. Lastly, using $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$ in the former case and $[\boldsymbol{t}_l]_{k-1}$ in the latter case, as well as the shares of the corrupted parties of $[\boldsymbol{r}_l]_{n-k}$ received from the adversary above, $\mathcal{S}$ can compute the corrupted parties' shares of each $[\boldsymbol{u}_l]_{n-k}$ and send them to $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$.

Now we argue that the real and ideal worlds are distributed identically. Let us begin with the case in which $P_1$ is corrupt. Since each $[\boldsymbol{r}_l]_{n-1}$ is random of degree-$(n-1)$, so too is $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-1}$, and therefore, the honest parties' shares in the real world are random, as the $t_l^i$ in the ideal world are. Since honest parties always send $t_l^i = u_l^i + r_l^i$ in Step 3, it must be that $u_l^i = t_l^i - r_l^i$. Therefore, the $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-t}$ that $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ reconstructs is the same as that defined by the corrupt $P_1$ in the real world. Finally, since $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ samples the sharings $[\boldsymbol{u}_l]_{n-k}$ based on the corrupted parties' shares, the output honest parties' shares are distributed identically in the two worlds.

Now, if $P_1$ is honest, it is clear that $\mathcal{S}$ computes the correct corrupted parties' shares of each $[\boldsymbol{u}_l]_{n-1}$ and thus the $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-t}$ that $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ reconstructs is the same as that defined by the corrupt parties' shares in the real world. Now, it is clear that the underlying $\boldsymbol{r}_l$ are unknown and random to the adversary. Thus, so too is the underlying $\boldsymbol{u}_l + \boldsymbol{r}_l$ and so $[\boldsymbol{t}_l]_{k-1}$ in the ideal world is distributed identically to that of $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{k-1}$ in the real world. Finally, as above, since $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ samples the sharings $[\boldsymbol{u}_l]_{n-k}$ based on the corrupted parties' shares, the output honest parties' shares are distributed identically in the two worlds. $\square$

**Triple sacrificing.** The triples produced by $\mathcal{F}_{\mathsf{triple}}$ may have some error. Thus, we present the following standard procedure $\pi_{\mathsf{sacrifice}}$, which takes two packed triples output from $\mathcal{F}_{\mathsf{triple}}$ that have been authenticated and sacrifices one of them to ensure that both have no error (including the other which is output). Note that although corrupted parties can change their shares of authenticated packed triples to change the underlying secrets (and thus add back error), they cannot do so without getting caught in the online phase, since the MAC key $\gamma$ is random and unknown to them.

**Procedure 9:** $\pi_{\mathsf{sacrifice}}$

1. All parties take as input packed triple sharings $([\![\boldsymbol{a}_1]\!]_{n-k}, [\![\boldsymbol{b}_1]\!]_{n-k}, [\![\boldsymbol{c}_1]\!]_{n-k})$ and $([\![\boldsymbol{a}_2]\!]_{n-k}, [\![\boldsymbol{b}_2]\!]_{n-k}, [\![\boldsymbol{c}_2]\!]_{n-k})$.
2. All parties invoke $\mathcal{F}_{\mathsf{coin}}$ and receive a random element $\rho \in \mathbb{F}$.
3. Then they compute $[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]_{n-k}$ and $[\boldsymbol{b}_2 - \boldsymbol{b}_1]_{n-k}$ and open them to $P_1$.
4. $P_1$ next reconstructs $\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1$ and $\boldsymbol{b}_2 - \boldsymbol{b}_1$, then distributes packed Shamir sharings $[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]_{k-1}$, $[\boldsymbol{b}_2 - \boldsymbol{b}_1]_{k-1}$, and $[(\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1) * (\boldsymbol{b}_2 - \boldsymbol{b}_1)]_{k-1}$.
5. Then, the parties compute $[\boldsymbol{\gamma} * \boldsymbol{\theta}]_{n-1} \leftarrow [\boldsymbol{\gamma}]_{n-k} * [(\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1) * (\boldsymbol{b}_2 - \boldsymbol{b}_1)]_{k-1} + [\boldsymbol{\gamma} * \boldsymbol{b}_1]_{n-k} * [\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]_{k-1} + \rho \cdot [\boldsymbol{\gamma} * \boldsymbol{a}_1]_{n-k} * [\boldsymbol{b}_2 - \boldsymbol{b}_1]_{k-1} + \rho \cdot [\boldsymbol{\gamma} * \boldsymbol{c}_1]_{n-k} - [\boldsymbol{\gamma} * \boldsymbol{c}_2]_{n-k}$.

*Communication complexity of* $\pi_{\mathsf{sacrifice}}$. We ignore calls to $\mathcal{F}_{\mathsf{coin}}$.

1. Step 3 costs $2n$ elements.
2. Step 4 costs $2n$ elements.

The above totals $4n$ elements, or $O(1)$ per individual slot.

**Checking that Many Sharings Share 0.** Lastly, we present a procedure $\pi_{\mathsf{check\text{-}zero}}$, which takes as input $M$ degree-$(n-1)$ packed Shamir sharings, $[\boldsymbol{\theta}_1]_{n-1}, \ldots, [\boldsymbol{\theta}_M]_{n-1}$ and ensures that each of them share underlying value $\mathbf{0}$. To do so, the parties use $\mathcal{F}_{\mathsf{coin}}$ to generate several random values, then take a random linear combination of the $[\boldsymbol{\theta}_l]_{n-1}$ defined by these values to obtain $[\boldsymbol{\theta}]_{n-1}$. The parties then use $\mathcal{F}_{\mathsf{commit}}$ to commit to their shares, then open the shares to each other and check whether the underlying secret is $\mathbf{0}$ or not. A minor subtlety is that the input sharings $[\boldsymbol{\theta}_l]_{n-1}$ may not be randomly distributed degree-$(n-1)$ packed Shamir sharings (if for example they are the multiplication of degree-$(n-k)$ and-$(k-1)$ packed Shamir sharings). Thus, the parties re-randomize $[\boldsymbol{\theta}]_{n-1}$ with a random degree-$(n-1)$ packed Shamir sharing $[\mathbf{0}]_{n-1}$, obtained from $\mathcal{F}_{\mathsf{rand}}$.

**Procedure 10:** $\pi_{\mathsf{check\text{-}zero}}$

1. All parties take as input $M$ degree-$(n-1)$ packed Shamir sharings, denoted by $[\boldsymbol{\theta}_1]_{n-1}, \ldots, [\boldsymbol{\theta}_M]_{n-1}$.
2. All parties invoke $\mathcal{F}_{\mathsf{coin}}$ and receive random values $\chi_1, \ldots, \chi_M$.
3. All parties invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare a random degree-$(n-1)$ packed Shamir sharing of $\mathbf{0}$, denoted by $[\mathbf{0}]_{n-1}$.
4. All parties locally compute $[\boldsymbol{\theta}]_{n-1} \leftarrow [\mathbf{0}]_{n-1} + \sum_{l=1}^{M} \chi_l \cdot [\boldsymbol{\theta}_l]_{n-1}$.
5. All parties invoke $\mathcal{F}_{\mathsf{commit}}$ to commit their shares of $[\boldsymbol{\theta}]_{n-1}$ to one another.
6. All parties open the commitments of the shares of $[\boldsymbol{\theta}]_{n-1}$ to one another and check whether it is a degree-$(n-1)$ packed Shamir sharing of $\mathbf{0}$. If not, all parties abort. If not, all parties abort.

The communication complexity of $\pi_{\mathsf{check\text{-}zero}}$ is $O(n^2)$. If $M = \Omega(n)$, this is $O(1)$ per underlying shared value.

Assume that for at least one of the input sharings $[\boldsymbol{\theta}_l]_{n-1}$ such that the underlying vector $\boldsymbol{\theta}_l \neq \mathbf{0}$, this underlying vector is random and unknown to the adversary. We now provide the following lemma which shows that if the above holds, then the adversary cannot force $[\boldsymbol{\theta}]_{n-1}$ to open to $\mathbf{0}$.

**Lemma 14.** *If $\exists l^* \in [M]$ such that $\boldsymbol{\theta}_{l^*} \neq \mathbf{0}$ and is random and unknown to the adversary, then the honest parties will abort with all-but-negligible probability at the end of $\pi_{\mathsf{check\text{-}zero}}$.*

*Proof.* First note that since the adversary uses $\mathcal{F}_{\mathsf{commit}}$ to commit to its shares, the shares are independent of the underlying value $\boldsymbol{\theta} = \sum_{l=1}^{M} \chi_l \cdot \boldsymbol{\theta}_l$. If the parties do not abort, then by definition $\boldsymbol{\theta} = \mathbf{0}$, so we have that $\mathbf{0} = \sum_{l=1}^{M} \chi_l \cdot \boldsymbol{\theta}_l$. Thus, we have that $\boldsymbol{\theta}_{l^*} = -(\sum_{l \in [M] \setminus \{l^*\}} \chi_l \cdot \boldsymbol{\theta}_l)/\chi_{l^*}$, which can only happen with negligible probability, since $\boldsymbol{\theta}_{l^*}$ is random and unknown to the adversary. $\square$

## D Missing Proofs

### D.1 Proof of Lemma 6

*Proof.* First we define the simulator $\mathcal{S}$, which initializes $\mathsf{bad\text{-}shr} \leftarrow 0$:

1. **Extraction Level 1**:
   (a) $\mathcal{S}$ first samples random $\boldsymbol{u}_i, \boldsymbol{v}_i \in \mathbb{F}^t$ on behalf of each honest party $P_i$. Note if $P_i$ is late corrupted, $\mathcal{S}$ can send the adversary these random vectors.
   (b) Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{coin}}$ by sampling random $r \in \mathbb{F}$ and and sending it to the adversary. It then interprets $r$ as bit string $\mathbf{R}$, and locally computes the matrix $\boldsymbol{M} \leftarrow \mathsf{Gen}(\mathbf{R})$.
   (c) Then, for every corrupted party $P_j$, let $\boldsymbol{u}_j$ be the first vector it inputs to $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ as party $P_A$ and $\boldsymbol{v}_j$ be the first vector it inputs to $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ as party $P_B$. For each $l \in [m]$, for every ordered pair $(P_i, P_j)$ such that $\boldsymbol{M}_l[i], \boldsymbol{M}_l[j] \neq 0$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ as follows:

– If $P_i$ and $P_j$ are both corrupted, then $\mathcal{S}$ receives from the adversary $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_j^i$ and returns them to the adversary.
– If $P_i$ is corrupted and $P_j$ is honest, $\mathcal{S}$ first receives $\overline{\boldsymbol{u}}_i$ and $\boldsymbol{\alpha}_i^j$ from the adversary, then computes $\boldsymbol{\beta}_j^i \leftarrow \overline{\boldsymbol{u}}_i * \boldsymbol{v}_j - \boldsymbol{\alpha}_i^j$, and returns $\boldsymbol{\alpha}_i^j$ to the adversary.
– If $P_i$ is honest and $P_j$ is corrupted, $\mathcal{S}$ first receives $\overline{\boldsymbol{v}}_j$ and $\boldsymbol{\beta}_j^i$ from the adversary, then computes $\boldsymbol{\alpha}_i^j \leftarrow \boldsymbol{u}_i * \overline{\boldsymbol{v}}_j - \boldsymbol{\beta}_j^i$ and returns $\boldsymbol{\beta}_j^i$ to the adversary.
– If both $P_i$ and $P_j$ are honest, $\mathcal{S}$ samples random $\boldsymbol{\beta}_j^i$ and sets $\boldsymbol{\alpha}_i^j \leftarrow \boldsymbol{u}_i * \boldsymbol{v}_j - \boldsymbol{\beta}_j^i$.
– In each case, $\mathcal{S}$ computes $\boldsymbol{\varepsilon}^{(i,j)} \leftarrow \boldsymbol{\alpha}_i^j + \boldsymbol{\beta}_j^i - \boldsymbol{u}_i * \boldsymbol{v}_j$.

(d) Then, for each honest party $P_i$, $\mathcal{S}$ splits $\boldsymbol{u}_i$, $\boldsymbol{v}_i$ and each $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ into length-$k$ vectors $\{\boldsymbol{u}_{i,w}\}_{w \in [t/k]}$, $\{\boldsymbol{v}_{i,w}\}_{w \in [t/k]}$, $\{\boldsymbol{\alpha}_{i,w}^j\}_{w \in [t/k]}$, $\{\boldsymbol{\beta}_{i,w}^j\}_{w \in [t/k]}$, respectively. $\mathcal{S}$ also splits each $\boldsymbol{\varepsilon}^{(i,j)}$ into $\{\boldsymbol{\varepsilon}_w^{(i,j)}\}_{w \in [t/k]}$.

(e) Then, for $w \in [\lfloor t/k \rfloor]$:

i. For each honest party, $\mathcal{S}$ computes random packed sharings $[\boldsymbol{u}_{i,w}]_{n-k}$, $[\boldsymbol{v}_{i,w}]_{n-k}$ and sends the adversary the corrupted parties' shares (Note if the adversary later corrupts more parties, $\mathcal{S}$ just sends the adversary their shares). $\mathcal{S}$ also receives from each corrupted party $P_j$, the honest parties' shares of $[\boldsymbol{u}_{j,w}]_{n-k}, [\boldsymbol{v}_{j,w}]_{n-k}$, and samples the rest of the sharings based on the honest parties' shares and $\boldsymbol{u}_{j,w}, \boldsymbol{v}_{j,w}$, respectively.

ii. $\mathcal{S}$ computes $([\boldsymbol{a}_{1,w}]_{n-k}, \ldots, [\boldsymbol{a}_{m,w}]_{n-k})$ and $([\boldsymbol{b}_{1,w}]_{n-k}, \ldots, [\boldsymbol{b}_{m,w}]_{n-k})$ using the above and $\boldsymbol{M}$.

iii. Then, for each $l \in [m]$:

A. For each honest party $P_i$ such that $\boldsymbol{M}_l[i] \neq 0$, $\mathcal{S}$ computes $\boldsymbol{c}_{l,w}^i \leftarrow \sum_{j:\boldsymbol{M}_l[j]\neq 0} \boldsymbol{\alpha}_{i,w}^j + \boldsymbol{\beta}_{i,w}^j$ then computes degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{c}_{l,w}^i]_{n-k}$ and sends the adversary the corrupted parties' shares (Note if the adversary later corrupts more parties, $\mathcal{S}$ just sends the adversary their shares). $\mathcal{S}$ also receives from each corrupted party $P_j$ such that $\boldsymbol{M}_l[j] \neq 0$, the honest parties' shares of $[\boldsymbol{c}_{l,w}^j]_{n-k}$ and samples the rest of the sharing based on the honest parties' shares and $\boldsymbol{c}_{l,w}^j$.

B. Finally, using the above, $\mathcal{S}$ can compute $[\boldsymbol{c}_{l,w}]_{n-k}$. $\mathcal{S}$ also computes $\boldsymbol{\varepsilon}_{l,w} \leftarrow \sum_{(i,j):\boldsymbol{M}_l[i]\neq 0 \neq \boldsymbol{M}_l[j]} \boldsymbol{\varepsilon}_w^{(i,j)}$.

2. **Extraction Level 2**: For each $w \in [\lfloor t/k \rfloor]$:

(a) For $l \in [(m-1)/2+1]$, $\tau \in [k]$, $\mathcal{S}$ implicitly views $a_{l,w}^\tau, b_{l,w}^\tau, c_{l,w}^\tau$ as the $l$-th evaluation points of polynomials $A_{\tau,w}(\cdot)$ of degree $(m-1)/2$, $B_{\tau,w}(\cdot)$ of degree $(m-1)/2$, $C_{\tau,w}(\cdot)$ of degree $m-1$, respectively; i.e., $A_{\tau,w}(l) = a_{l,w}^\tau$, $B_{\tau,w}(l) = b_{l,w}^\tau$, and $C_{\tau,w}(l) = c_{l,w}^\tau$.

(b) Letting $\boldsymbol{A}_w(\cdot) \leftarrow (A_{1,w}(\cdot), \ldots, A_{k,w}(\cdot))$ and $\boldsymbol{B}_w(\cdot) \leftarrow (B_{1,w}(\cdot), \ldots, B_{k,w}(\cdot))$, $\mathcal{S}$ uses $([\boldsymbol{A}_w(1)]_{n-k}, \ldots, [\boldsymbol{A}_w((m-1)/2+1)]_{n-k})$ and $([\boldsymbol{B}_w(1)]_{n-k}, \ldots, [\boldsymbol{B}_w((m-1)/2+1)]_{n-k})$, which implicitly define the polynomials $\boldsymbol{A}_w(\cdot)$ and $\boldsymbol{B}_w(\cdot)$, respectively, to locally compute the corrupted parties' shares of $[\boldsymbol{A}_w(l)]_{n-k}$ and $[\boldsymbol{B}_w(l)]_{n-k}$ for $l \in [(m-1)/2+2, m]$.

(c) Then, for $l \in [(m-1)/2+2, m]$, to simulate $\pi_{\mathsf{beaver}}$, $\mathcal{S}$ first sets $\boldsymbol{\eta}_{l,w} \leftarrow \boldsymbol{0}$ and computes $\boldsymbol{d} \leftarrow \boldsymbol{A}_w(l) - \boldsymbol{a}_{l,w}$ and $\boldsymbol{e} \leftarrow \boldsymbol{B}_w(l) - \boldsymbol{b}_{l,w}$, then:

- If $P_1$ is corrupt:
  - i. $\mathcal{S}$ first locally computes the honest parties' shares of $[\boldsymbol{A}_w(l)]_{n-k} - [\boldsymbol{a}_{l,w}]_{n-k}$ and $[\boldsymbol{B}_w(l)]_{n-k} - [\boldsymbol{b}_{l,w}]_{n-k}$ then sends them to the adversary.
  - ii. $\mathcal{S}$ then receives from the adversary $(d_{i_1}, e_{i_1}), \ldots, (d_{i_{n-t}}, e_{i_{n-t}})$ on behalf of the honest parties. If these shares do not lie on a polynomial of degree $k-1$, $\mathcal{S}$ sets $\mathsf{bad\text{-}shr} \leftarrow 1$. Otherwise, it reconstructs $[\overline{\boldsymbol{d}}]_{k-1}, [\overline{\boldsymbol{e}}]_{k-1}$.
  - iii. In the latter case, $\mathcal{S}$ for each corrupted party then computes its share of $[\boldsymbol{C}_w(l)]_{n-1}$ as $[\boldsymbol{d}]_{k-1} * [\boldsymbol{e}]_{k-1} + [\boldsymbol{d}]_{k-1} * [\boldsymbol{b}_{l,w}]_{n-k} + [\boldsymbol{e}]_{k-1} * [\boldsymbol{a}_{l,w}]_{n-k} + [\boldsymbol{c}_{l,w}]_{n-k}$.
- If $P_1$ is honest:
  - i. $\mathcal{S}$ receives from the adversary the corrupt parties' shares of $[\boldsymbol{A}_w(l)]_{n-k} - [\boldsymbol{a}_{l,w}]_{n-k}$ and $[\boldsymbol{B}_w(l)]_{n-k} - [\boldsymbol{b}_{l,w}]_{n-k}$.
  - ii. Then, together with the honest parties' shares, $\mathcal{S}$ reconstructs $\overline{\boldsymbol{d}} \leftarrow \boldsymbol{A}_w(l) - \boldsymbol{a}_{l,w}$ and $\overline{\boldsymbol{e}} \leftarrow \boldsymbol{B}_w(l) - \boldsymbol{b}_{l,w}$, computes $[\overline{\boldsymbol{d}}]_{k-1}$ and $[\overline{\boldsymbol{e}}]_{k-1}$, and then sends the corrupt parties' their shares.
  - iii. Finally, $\mathcal{S}$ computes the corrupted parties' shares of $[\boldsymbol{C}_w(l)]_{n-1}$ as above.

In both cases, $\mathcal{S}$ computes $\boldsymbol{\eta}_{l,w} \leftarrow (\boldsymbol{d}_{l,w} * \boldsymbol{e}_{l,w} + \boldsymbol{d}_{l,w} * \boldsymbol{b}_{l,w} + \boldsymbol{e}_{l,w} * \boldsymbol{a}_{l,w} + \boldsymbol{c}_{l,w}) - (\overline{\boldsymbol{d}}_{l,w} * \overline{\boldsymbol{e}}_{l,w} + \overline{\boldsymbol{d}}_{l,w} * \boldsymbol{b}_{l,w} + \overline{\boldsymbol{e}}_{l,w} * \boldsymbol{a}_{l,w} + \boldsymbol{c}_{l,w})$

(d) Note: sharings $([\boldsymbol{A}_w(1)]_{n-k}, \ldots, [\boldsymbol{A}_w(m)]_{n-k})$, $([\boldsymbol{B}_w(1)]_{n-k}, \ldots, [\boldsymbol{B}_w(m)]_{n-k})$, and $([\boldsymbol{C}_w(1)]_{n-1}, \ldots, [\boldsymbol{C}_w(m)]_{n-1})$ implicitly define the polynomials $\boldsymbol{A}_w(\cdot)$, $\boldsymbol{B}_w(\cdot)$, and $\boldsymbol{C}_w(\cdot)$, respectively. Thus, for $l \in [m+1, , m+\gamma \cdot m-(m+1)/2]$, $\mathcal{S}$ locally computes the corrupted parties' shares of

$$([\boldsymbol{a}^{l,w}]_{n-k}, [\boldsymbol{b}^{l,w}]_{n-k}, [\boldsymbol{c}^{l,w}]_{n-1}) \leftarrow ([\boldsymbol{A}_w(m+l)]_{n-k}, [\boldsymbol{B}_w(m+l)]_{n-k}, [\boldsymbol{C}_w(m+l)]_{n-1}).$$

The errors $\boldsymbol{\varepsilon}_{1,w}, \ldots, \boldsymbol{\varepsilon}_{(m-1)/2+1}, \boldsymbol{\varepsilon}_{(m-1)/2+2} + \boldsymbol{\eta}_{(m-1)/2+2}, \ldots, \boldsymbol{\varepsilon}_m + \boldsymbol{\eta}_m$ also define degree-$m-1$ polynomials $\boldsymbol{\delta}_w(\cdot)$. $\mathcal{S}$ computes $\boldsymbol{\delta}_w(l)$ for $l \in [m+1, m+\gamma \cdot m-(m+1)/2]$ based on these.

3. **Checking the degree of $P_1$'s sharings**: $\mathcal{S}$ then emulates $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ by first sending the adversary the corrupted parties' shares of the sharings $[\overline{\boldsymbol{d}}]_{d_1}, [\overline{\boldsymbol{e}}]_{d_2}$ from $\pi_{\mathsf{beaver}}$, then sending $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{triple}}$ if $\mathsf{bad\text{-}shr} = 1$ or the adversary sends $\mathsf{abort}$ to $\mathcal{S}$.
4. **Output**: If $\mathcal{S}$ has not yet aborted it sends to $\mathcal{F}_{\mathsf{triple}}$ the corrupted parties' shares of $([\boldsymbol{a}^{l,w}]_{n-k}, [\boldsymbol{b}^{l,w}]_{n-k}, [\boldsymbol{c}^{l,w}]_{n-1})$ and $\boldsymbol{\delta}_w(l)$ for all $w \in [t/k], l \in [m+1, m+\gamma \cdot m-(m+1)/2]$.

Now we must show that the real and ideal worlds are indistinguishable. It is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{coin}}$ and $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ in the first extraction level as in the real world, and that $\boldsymbol{\delta}^{(i,j)}$ correctly computes the difference between $\boldsymbol{u}_i * \boldsymbol{v}_j$ based on the first seeds that the corrupted parties input to $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ and those that it inputs to the $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ invocation for the ordered pair $(i,j)$. It is also clear that the distribution of corrupted parties' shares of honest sharings $[\boldsymbol{u}_{i,w}]_{n-k}, [\boldsymbol{v}_{i,w}]_{n-k}$ are identical in the two worlds, and same with that of $[\boldsymbol{c}^i_{l,w}]_{n-k}$.

For the simulation of $\pi_{\mathsf{beaver}}$ in the second extraction level, let us first handle the case in which $P_1$ is corrupt. Since $\mathcal{S}$ just uses the already computed honest parties' shares when sending to $P_1$, $[\boldsymbol{A}_w(l)]_{n-k} - [\boldsymbol{a}_{l,w}]_{n-k}$ and $[\boldsymbol{B}_w(l)]_{n-k} - [\boldsymbol{b}_{l,w}]_{n-k}$, it is clear that this is an identical distribution to that of the real world. It is also clear that $\mathcal{S}$ correctly sets $\mathsf{bad\text{-}shr} \leftarrow 1$ if the honest parties' shares of $[\boldsymbol{d}]_{k-1}, [\boldsymbol{e}]_{k-1}$ do not lie on a degree-$(k-1)$ polynomial. Now, if $P_1$ is honest, again since $\mathcal{S}$ just uses the already computed honest parties' shares, the reconstruction of $\boldsymbol{d}$ and $\boldsymbol{e}$ occur as in the real world, as does the computation of $[\boldsymbol{d}]_{k-1}, [\boldsymbol{e}]_{k-1}$. It is also clear that $\mathcal{S}$'s emulation of $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ is identical to the real world.

Now we show that if the honest parties do not abort, then their shares are distributed identically in the ideal and real worlds. Let $H \subseteq \mathsf{Hon}$ be some subset of $\mathsf{Hon}$ of size $|H| = n - t$ and $\boldsymbol{M}^H$ be the sub-matrix of $\boldsymbol{M}$ that contains the columns corresponding to the indices of the honest parties in $H$. From Theorem 4, we have that with all-but-negligible probability, $\boldsymbol{M}^H$ has rank at least $\gamma \cdot m$, for $\gamma > 1/2$. This means that there must be some $(\gamma \cdot m) \times (\gamma \cdot m)$ dimensional sub-matrix $\boldsymbol{M}'$ of $\boldsymbol{M}^H$ that is invertible. Let $R = \{r_1, \dots r_{\gamma \cdot m}\}$ be the rows of $\boldsymbol{M}$ of which $\boldsymbol{M}'$ consists, $C = \{c_1, \dots, c_{\gamma \cdot m}\}$ be those columns, and $\overline{C} = \{\overline{c}_1, \dots, \overline{c}_{n-\gamma \cdot m}\}$ be those columns of $\boldsymbol{M}$ that are not in $\boldsymbol{M}'$. Then we can write

$$\begin{aligned}([\boldsymbol{a}_{r_1,w}]_{n-k}, \dots [\boldsymbol{a}_{r_{\gamma\cdot m},w}]_{n-k})^{\intercal} =& \boldsymbol{M}' \cdot ([\boldsymbol{u}_{c_1,w}]_{n-k}, \dots [\boldsymbol{u}_{c_{\gamma \cdot m},w}]_{n-k})^{\intercal} + \\ & \boldsymbol{M}_R^{\overline{C}} \cdot ([\boldsymbol{u}_{\overline{c}_1,w}]_{n-k}, \dots [\boldsymbol{u}_{\overline{c}_{\gamma\cdot m},w}]_{n-k})^{\intercal}.\end{aligned}$$

Since $\boldsymbol{M}'$ is invertible, for any fixed sharings $([\boldsymbol{u}_{\overline{c}_1,w}]_{n-k}, \dots [\boldsymbol{u}_{\overline{c}_{\gamma\cdot m},w}]_{n-k})$, the distribution of $([\boldsymbol{a}_{r_1,w}]_{n-k}, \dots [\boldsymbol{a}_{r_{\gamma\cdot m},w}]_{n-k})$ is the same as $([\boldsymbol{u}_{c_1,w}]_{n-k}, \dots [\boldsymbol{u}_{c_{\gamma\cdot m},w}]_{n-k})$, which is random given the shares of corrupted parties. Note also that what the adversary receives from $\mathcal{S}$'s emulation of $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ is independent of the $\boldsymbol{u}_{c_1,w}, \dots, \boldsymbol{u}_{c_{\gamma\cdot m},w}$. Furthermore, the distribution of the sharings $[\boldsymbol{c}^i_{l,w}]$ for each honest party $P_i$ is random given the corrupted parties' shares. So, the above still holds that the distribution of $([\boldsymbol{a}_{r_1,w}]_{n-k}, \dots [\boldsymbol{a}_{r_{\gamma\cdot m},w}]_{n-k})$ is random given the shares of corrupted parties.

In the second level of extraction, consider the vector of polynomials $\boldsymbol{A}_w(\cdot)$. We have two cases. First, we have the case that $[(m-1)/2+1] \subseteq R$. In this case, we have from above that for each $r \in R \cap [(m-1)/2+1] = [(m-1)/2+1]$, the sharing $[\boldsymbol{A}_w(r)]_{n-k} = [\boldsymbol{a}_{r,w}]_{n-k}$ is distributed randomly given the shares of corrupted parties. Also, for each $r \in R \cap [(m-1)/2+2, m]$, the sharing $[\boldsymbol{a}_{r,w}]_{n-k}$ is distributed randomly given the shares of corrupted parties, and thus so is $[\boldsymbol{A}_w(r)]_{n-k} + [\boldsymbol{a}_{r,w}]_{n-k}$. Therefore, all but $m - \gamma \cdot m$ points of each polynomial $\boldsymbol{A}_w(\cdot)$ are random to the adversary. Indeed, since each polynomial in $\boldsymbol{A}_w$ is of degree $(m-1)/2$, the $m - \gamma \cdot m < (m-1)/2 + 1$ points of each polynomial which may not be random to the adversary are independent of the remaining

$(m-1)/2+1-(m-\gamma\cdot m)=\gamma\cdot m-(m+1)/2$ points, all of which are random. This means that the sharings $[\boldsymbol{A}_w(r)]_{n-k}$ corresponding to these points are still distributed randomly given the shares of the corrupted parties.

In the second case, we have that $[(m-1)/2+1]\nsubseteq R$, which means that $[(m-1)/2+2,m]\subseteq R$, since $|R|=\gamma\cdot m>m/2$. In this case, we have from above that for each $r\in R\cap[(m-1)/2+1]$, the sharing $[\boldsymbol{A}_w(r)]_{n-k}=[\boldsymbol{a}_{r,w}]_{n-k}$ is distributed randomly given the shares of corrupted parties. Moreover, since each polynomial in $\boldsymbol{A}_w$ is of degree $(m-1)/2$, the points of each polynomial corresponding to $[(m-1)/2+1]\setminus R$ are independent of those that are in $R\cap[(m-1)/2+1]$. Furthermore, for each $r\in R\cap[(m-1)/2+2,m]=[(m-1)/2+2,m]$, the sharing $[\boldsymbol{a}_{r,w}]_{n-k}$ is distributed randomly given the shares of corrupted parties, and thus so is $[\boldsymbol{A}_w(r)]_{n-k}+[\boldsymbol{a}_{r,w}]_{n-k}$. Therefore, the adversary learns nothing else about those $[\boldsymbol{A}_w(r)]_{n-k}$ for $r\in R\cap[(m-1)/2+1]$, of which there must be $(m-1)/2+1-(m-\gamma\cdot m)=\gamma\cdot m-(m+1)/2$.

In either case, $[\boldsymbol{A}_w(1)]_{n-k},\ldots,[\boldsymbol{A}_w((m-1)/2+1)]_{n-k}$, of which $\gamma\cdot m-(m+1)/2$ are random given the corrupted parties' shares, are used to compute the output $[\boldsymbol{A}_w(m+l)]_{n-k}$ for $l\in[m+1,m+\gamma\cdot m-(m+1)/2]$, in a one-to-one fashion (since the computation is represented by a super-invertible matrix whose number of rows is equal to $\gamma\cdot m-(m+1)/2$), which are therefore random given the corrupted parties' shares.

In the ideal world, $\mathcal{S}$ generates the corrupted parties' shares of those $[\boldsymbol{u}_{j,w}]_{n-k}$ for $j\in\mathsf{Corr}$ by randomly sampling their shares based on those of the honest parties and the underlying $\boldsymbol{u}$, which $\mathcal{S}$ received from its emulation of $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$. Using these, and the corrupted parties' shares of honest parties' sharings, $\mathcal{S}$ computes the corrupted parties' shares of the output sharings by following the protocol description. Then the shares of the honest parties are randomly sampled based on the shares of corrupted parties. Thus the distribution of the output shares of honest parties are identical in the two worlds.

Note that the same exact argument as above can be used for the output $[\boldsymbol{B}_w(m+l)]_{n-k}$ for $l\in[m+1,m+\gamma\cdot m-(m+1)/2]$.

For the honest parties' shares of the output $[\boldsymbol{c}^{l,w}]_{n-k}$, recall that the distribution of the sharings $[\boldsymbol{c}^i_{l,w}]$ for each honest party $P_i$ is random given the corrupted parties' shares. Since the $[\boldsymbol{c}^{l,w}]_{n-k}$ are linear combinations of $(([\boldsymbol{a}_{1,w}]_{n-k},[\boldsymbol{b}_{1,w}]_{n-k},[\boldsymbol{c}_{1,w}]_{n-k}),\ldots,([\boldsymbol{a}_{m,w}]_{n-k},[\boldsymbol{b}_{m,w}]_{n-k},[\boldsymbol{c}_{m,w}]_{n-k}))$, then it must be that the shares of honest parties are random given the corrupted parties' shares and such that they are consistent with $\boldsymbol{a}^{l,w}*\boldsymbol{b}^{l,w}$ and any errors which the adversary injected for $\mathcal{F}_{\mathsf{OLE}}^{\mathbf{prog}}$ and in $\pi_{\mathsf{beaver}}$. Since $\mathcal{S}$ tracks these errors and inputs them to $\mathcal{F}_{\mathsf{triple}}$, the ideal world is distributed identically to the real world. □

## D.2 Proof of Lemma 7

*Proof.* First we define the simulator $\mathcal{S}$, which begins by setting $\mathsf{bad\text{-}shr}\leftarrow 0$:

1. $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{triple}}$ by receiving from the adversary the corrupted parties' shares and $(\boldsymbol{\delta}_{l,w})_{l\in\mu,w\in[\lfloor t/k\rfloor]}$. It then samples random $([\boldsymbol{a}_{l,w}]_{n-k},[\boldsymbol{b}_{l,w}]_{n-k},[\boldsymbol{a}_{l,w}*\boldsymbol{b}_{l,w}+\delta_{l,w}]_{n-k})_{l\in\mu,w\in[\lfloor t/k\rfloor]}$ consistent with the corrupted parties' shares. Note,

if the adversary corrupts more parties after the invocation of $\mathcal{F}_{\mathsf{triple}}$, it can just send the corresponding shares to the adversary.

2. Then for $w \in [\lfloor t/k \rfloor], l \in [\mu]$:
   (a) If $P_1$ is corrupted, $\mathcal{S}$ sends the adversary random $\rho^i_{l,w}$ for $i \in \mathsf{Hon}$. Else it receives from the adversary its shares of $[\boldsymbol{\gamma} + \boldsymbol{s}_{l,w}]_{n-k}$ and using the corrupted parties' shares of $[\boldsymbol{s}_{l,w}]_{n-k}$ received from the adversary above, $\mathcal{S}$ computes their shares of $[\boldsymbol{\gamma}]_{n-k}$ and sends them to $\mathcal{F}_{\mathsf{auth\text{-}rand}}$.
   (b) Then, if $P_1$ is corrupted, $\mathcal{S}$ receives from the adversary the honest parties' shares of $[\boldsymbol{\gamma}+\boldsymbol{s}_{l,w}]_{k-1}$. If these shares do not lie on a polynomial of degree $k-1$, $\mathcal{S}$ sets $\mathsf{bad\text{-}shr} \leftarrow 1$. Otherwise, it reconstructs $\boldsymbol{\gamma} + \boldsymbol{s}_{l,w}$ and since $\boldsymbol{s}_{l,w}$ is sampled above, computes $\boldsymbol{\gamma}$. Then, $\mathcal{S}$ can compute for each honest party $\gamma^i \leftarrow \rho^i_{l,w} - s^i_{l,w}$ and based on these and $\boldsymbol{\gamma}$, sample a sharing $[\boldsymbol{\gamma}]_{n-k}$ and send the corrupted parties' shares to $\mathcal{F}_{\mathsf{auth\text{-}rand}}$. If $P_1$ is honest, $\mathcal{S}$ samples random sharing $[\boldsymbol{\rho}_{l,w}]_{k-1}$ and sends the corrupted parties their shares.
   (c) Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ by receiving from the adversary the corrupted parties' shares of $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-1}$ and their shares of $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}-\boldsymbol{\delta}_{l,w}]_{n-k}$. $\mathcal{S}$ can also compute the honest parties' shares of the former and use them together with the corrupted parties' shares to reconstruct $\boldsymbol{\gamma} * \boldsymbol{r}_{l,w} - \boldsymbol{\delta}_{l,w}$, then compute and sample random sharing $[\boldsymbol{\gamma} * \boldsymbol{r}_{l,w} - \boldsymbol{\delta}_{l,w}]_{n-k}$ consistent with the corrupted parties' shares. Note that at this point, (if it hasn't already) $\mathcal{S}$ can also compute $\boldsymbol{\gamma}$ and use it along with the corrupted parties' shares of $[\boldsymbol{\gamma}]_{n-k}$ to sample the rest of the sharing.
3. Next, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ by by first sending the adversary the corrupted parties' shares of the sharings $[\boldsymbol{\gamma} + \boldsymbol{s}_{l,w}]_{k-1}$ then aborting if $\mathsf{bad\text{-}shr} = 1$.
4. Finally, $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ the corrupted parties' shares of $[\boldsymbol{r}_{l,w}]_{n-k}$. As for $[\boldsymbol{\gamma} * \boldsymbol{r}_{l,w} - \boldsymbol{\delta}_{l,w}]_{n-k}$, first note that the corrupted parties can locally change their shares to change the secret to $[\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}]_{n-k}$. We may equivalently think that the shares of corrupted parties are changed so that the secret is $\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}$ (and they can then change their shares to any arbitrary values). Thus, $\mathcal{S}$ adjusts the shares of corrupted parties as follows: $\mathcal{S}$ generates a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\delta}_{l,w}]_{n-k}$ such that the shares of honest parties are 0's. Then $\mathcal{S}$ computes the shares of $[\boldsymbol{\gamma}*\boldsymbol{r}_{l,w}]_{n-k} \leftarrow [\boldsymbol{\gamma}*\boldsymbol{r}_{l,w} - \boldsymbol{\delta}_{l,w}]_{n-k} + [\boldsymbol{\delta}_{l,w}]_{n-k}$ of the corrupted parties and sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ these shares.

Now, we show that the real and ideal worlds are distributed identically, except with all-but-negligible probability. It is clear that $\mathcal{S}$'s emulation of $\mathcal{F}_{\mathsf{triple}}$ is identical to the real world. If $P_1$ is corrupted, then $\mathcal{S}$ sending the adversary random values for the opening of $[\boldsymbol{\gamma} + \boldsymbol{s}_{l,w}]_{n-k}$ is distributed identically to the real world since $[\boldsymbol{s}_{l,w}]_{n-k}$ is distributed randomly, given the corrupted parties shares. If $P_1$ is honest, it is clear that it can extract the correct corrupted parties' shares of $[\boldsymbol{\gamma}]_{n-k}$. If $P_1$ is corrupted, then $\mathcal{S}$ can clearly reconstruct $\boldsymbol{\gamma} + \boldsymbol{s}_{l,w}$ and from this extract $\boldsymbol{\gamma}$ then, along with the computed honest parties' shares, sample a sharing $[\boldsymbol{\gamma}]_{n-k}$ that is consistent with the corrupted parties' shares. If $P_1$ is honest, it is clear that simulated $[\boldsymbol{\rho}]_{k-1}$ is identical to the real world, since $[\boldsymbol{s}_{l,w}]_{n-k}$ is distributed randomly, given the corrupted parties shares, and so then

$\boldsymbol{s}_{l,w}$ is random to the adversary. Then, it is clear that $\mathcal{S}$'s emulation of $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ is identical to the real world and also, that $\mathcal{S}$ can compute $\boldsymbol{\gamma}$ and, along with the corrupted parties' shares, sample $[\boldsymbol{\gamma}]_{n-k}$.

Lastly, we need to show that the honest parties shares of $[\![\boldsymbol{r}_{l,w}]\!]$ are distributed identically in both worlds. We again have that the corrupted parties' shares of $[\boldsymbol{r}_{l,w}]_{n-k}$ are independent of the rest of the sharings. Since $\mathcal{S}$ also has the corrupted parties' shares, this means that the sharings $[\boldsymbol{r}_{l,w}]$ that $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ samples are distributed identically to those in the real world. $\mathcal{S}$ also gets the corrupted parties' shares of $[\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}]$ from its emulation of $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$, which are also independent of the rest of the sharings by the definition of $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$. Thus, $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ samples $[\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}]$ distributed identically to those in the real world, where in the real world, the corrupted parties are also changed so that the secret is $\boldsymbol{\gamma} * \boldsymbol{r}_{l,w}$. □

## D.3 Proof of Theorem 6

*Proof.* We will construct a simulator $\mathcal{S}$. First, we describe how $\mathcal{S}$ simulates $\pi_{\mathsf{auth}}$, with knowledge of the whole sharing $[\boldsymbol{\gamma}]_{n-k}$ and each $([\boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{u}_{n-t}]_{n-k})$, which starts by setting $\mathsf{bad\text{-}shr} \leftarrow 0$:

1. $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ by receiving from the adversary the corrupted parties' shares of $[\boldsymbol{\gamma}]_{n-k}$ and then either $\mathsf{abort}$ or the corrupted parties' shares of each $[\![\boldsymbol{r}_l]\!]_{n-k}$. In the former case, $\mathcal{S}$ sends $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$; otherwise, it samples random sharings $[\![\boldsymbol{r}_l]\!]_{n-k} = ([\boldsymbol{r}_l]_{n-k}, [(\boldsymbol{\gamma} + \boldsymbol{\varepsilon}) * \boldsymbol{r}_l]_{n-k})$ consistent with the corrupted parties' shares, where $\boldsymbol{\varepsilon}$ is the difference between $\boldsymbol{\gamma}$ and that which is defined by the honest parties' shares of $[\boldsymbol{\gamma}]_{n-k}$ and those input by the adversary above. Note, if the adversary corrupts more parties later, it can just send their shares to the adversary.
2. Then for $l \in [n-t]$:
   (a) If $P_1$ is corrupted, $\mathcal{S}$ sends to the adversary the honest parties' shares of $[\boldsymbol{u}_l + \boldsymbol{r}_l]_{n-k}$. Otherwise, it receives from the adversary the corrupted parties' shares of $[\boldsymbol{u}_l + \boldsymbol{r}_l + \boldsymbol{\eta}_l]_{n-k}$, where $\boldsymbol{\eta}_l$ is error added by the adversary.
   (b) Then, if $P_1$ is corrupted, $\mathcal{S}$ receives from the adversary $[\boldsymbol{u}_l + \boldsymbol{r}_l + \boldsymbol{\eta}_l]_{k-1}$, where $\boldsymbol{\eta}_l$ is error added by the adversary. If these shares do not lie on a polynomial of degree $k-1$, $\mathcal{S}$ sets $\mathsf{bad\text{-}shr} \leftarrow 1$. If $P_1$ is honest, $\mathcal{S}$ computes sharing $[\boldsymbol{u}_l + \boldsymbol{r}_l + \boldsymbol{\eta}_l]_{k-1}$ and sends the corrupted parties their shares.
   (c) In either case, $\mathcal{S}$ can compute the sharing $[\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-1} \leftarrow [\boldsymbol{\gamma}]_{n-k} * [\boldsymbol{u}_l + \boldsymbol{r}_l + \boldsymbol{\eta}_l]_{k-1} - [(\boldsymbol{\gamma} + \boldsymbol{\varepsilon}) * \boldsymbol{r}_l]_{n-k}$
3. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ by receiving from the adversary the corrupted parties' shares of each $[\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-1}$ and then of each $[\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-k}$, and then randomly sampling the rest of the latter sharings, consistent with the corrupted parties' shares and the values $\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l$.
4. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ by sending the adversary the corrupted parties' shares of each $[\boldsymbol{u}_l + \boldsymbol{r}_l + \boldsymbol{\eta}_l]_{k-1}$ and then sending $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ $\mathsf{abort}$ if $\mathsf{bad\text{-}shr} = 1$ or the adversary sends $\mathcal{S}$ $\mathsf{abort}$.
5. If $\mathcal{S}$ does not abort, it outputs the sharings $([\boldsymbol{u}_l]_{n-k}, [\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-k})$.

If in fact it is the case that the sharings $[\boldsymbol{\gamma}]_{n-k}$ and each $([\boldsymbol{u}_1]_{n-k}, \ldots, [\boldsymbol{u}_{n-t}]_{n-k})$ are distributed identically to the real world, then we show that $\mathcal{S}$'s simulation of $\pi_{\mathsf{auth}}$ is identical to the real world. It is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ exactly as in the real world. It is also clear that $\mathcal{S}$ simulates the opening and redistribution of $[\boldsymbol{u}_l + \boldsymbol{r}_l + \boldsymbol{\eta}_l]_{k-1}$ as in the real world. Finally, it is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ and $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ as in the real world.

Now, we also note that the corrupted parties can locally change their shares to change the secret from $\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l$ to $\boldsymbol{\gamma} * \boldsymbol{u}_l$. We may equivalently think that the shares of corrupted parties are changed so that the secret is $\boldsymbol{\gamma} * \boldsymbol{u}_l$ (and they can then change their shares to any arbitrary values). Thus, $\mathcal{S}$ adjusts the shares of corrupted parties as follows: $\mathcal{S}$ generates a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\gamma} * \boldsymbol{\eta}_l - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-k}$ such that the shares of honest parties are 0's. Then $\mathcal{S}$ computes the sharing $[\boldsymbol{\gamma} * \boldsymbol{u}_l]_{n-k} \leftarrow [\boldsymbol{\gamma} * (\boldsymbol{u}_l + \boldsymbol{\eta}_l) - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-k} - [\boldsymbol{\gamma} * \boldsymbol{\eta}_l - \boldsymbol{\varepsilon} * \boldsymbol{r}_l]_{n-k}$ and uses this sharing in the rest of the simulation.

Now, we describe the rest of $\mathcal{S}$:

1. For sampling the MAC key, $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary the corrupted parties' shares of $[\boldsymbol{\gamma}]_{n-k}$ and then sampling the rest of the sharing randomly, consistent with the corrupted parties' shares. $\mathcal{S}$ then sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ the corrupted parties' shares. Note that if the adversary later corrupts more parties, $\mathcal{S}$ can just send their shares to the adversary.
2. **Preparing Random, Authenticated Sharings for Input and Output Gates**:
   (a) $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{triple}}$ by receiving from the adversary the corrupted parties' shares of each $[\boldsymbol{\Delta}_l]_{n-k}$, $[\boldsymbol{r}_l]_{n-k}$, and $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-k}$, and each $\boldsymbol{\delta}_l$, and then sampling the rest of the sharings randomly, consistent with the corrupted parties' shares and each $\boldsymbol{\delta}_l$. Note that if the adversary later corrupts more parties, $\mathcal{S}$ can just send their shares to the adversary.
   (b) Then, $\mathcal{S}$ simulates $\pi_{\mathsf{auth}}$ as above, using the whole sharings $[\boldsymbol{\gamma}]_{n-k}$ and $[\boldsymbol{r}_l]_{n-k}$.
   (c) Next, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ by receiving from the adversary the corrupted parties' shares of each $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-1}$ and the new sharings $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-k}$, and then randomly sampling the latter consistent with the corrupted parties' share and the underlying values.
   (d) Finally, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary the corrupted parties' shares of $[\mathbf{0}]_{n-1}$ and randomly sampling the rest of the sharing based on the corrupted parties' shares and $\mathbf{0}$.

   $\mathcal{S}$ then sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ the corrupted parties' shares of each $[\![\boldsymbol{r}_l]\!]_{n-k}$, $[\boldsymbol{\Delta}_l]_{n-k}$, and $[\mathbf{0}]_{n-1}$. As for $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-k}$, first note that the corrupted parties can locally change their shares to change the secret to $\boldsymbol{\Delta}_l * \boldsymbol{r}_l$. We may equivalently think that the shares of corrupted parties are changed so that the secret is $\boldsymbol{\Delta}_l * \boldsymbol{r}_l$ (and they can then change their shares to any arbitrary values). Thus, $\mathcal{S}$ adjusts the shares of corrupted parties as follows: $\mathcal{S}$ generates a degree-$(n-k)$ packed Shamir sharing $[\boldsymbol{\delta}_l]_{n-k}$ such that the shares of honest parties are 0's. Then $\mathcal{S}$ computes the shares of $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l]_{n-k} \leftarrow [\boldsymbol{\Delta}_l * \boldsymbol{r}_l + \boldsymbol{\delta}_l]_{n-k} - [\boldsymbol{\delta}_l]_{n-k}$ of the corrupted parties and sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ these shares.

3. **Preparing Packed, Authenticated Beaver Triples**:
   (a) $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{triple}}$ by receiving from the adversary the corrupted parties' shares of each $[\boldsymbol{a}_l]_{n-k}$, $[\boldsymbol{b}_l]_{n-k}$, and $[\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-k}$, and each $\boldsymbol{\delta_l}$, and then sampling the rest of the sharings randomly, consistent with the corrupted parties' shares and each $\boldsymbol{\delta_l}$. Note that if the adversary later corrupts more parties, $\mathcal{S}$ can just send their shares to the adversary.
   (b) Then, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ by receiving from the adversary the corrupted parties' shares of each $[\boldsymbol{c}_l * + \boldsymbol{\delta}_l]_{n-1}$ and the new sharings $[\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-k}$, and then randomly sampling the latter consistent with the corrupted parties' share and the underlying values.
   (c) Then, $\mathcal{S}$ simulates $\pi_{\mathsf{auth}}$ as above, using the whole sharings $[\boldsymbol{\gamma}]_{n-k}$ and $[\boldsymbol{a}_l]_{n-k}$, $[\boldsymbol{b}_l]_{n-k}$, $[\boldsymbol{c}_l + \boldsymbol{\delta}_l]_{n-k}$
   (d) To simulate $\pi_{\mathsf{sacrifice}}$:
      i. $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{coin}}$ by sampling random $\rho$ and sending it to the adversary.
      ii. If $P_1$ is corrupted, $\mathcal{S}$ sends to the adversary the honest parties' shares of $[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]_{n-k}$ and $[\boldsymbol{b}_2 - \boldsymbol{b}_1]_{n-k}$. Otherwise, it receives from the adversary the corrupted parties' shares of $\overline{[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]}_{n-k}$ and $\overline{[\boldsymbol{b}_2 - \boldsymbol{b}_1]}_{n-k}$.
      iii. Then, if $P_1$ is corrupted, $\mathcal{S}$ receives from the adversary $\overline{[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]}_{k-1}$, $\overline{[\boldsymbol{b}_2 - \boldsymbol{b}_1]}_{k-1}$, and $\overline{[(\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1) * (\boldsymbol{b}_2 - \boldsymbol{b}_1)]}_{k-1}$. If the shares do not lie on a polynomial of degree $k-1$ for any of the sharings, $\mathcal{S}$ sets $\mathsf{bad\text{-}shr} \leftarrow 1$. If $P_1$ is honest, $\mathcal{S}$ computes the sharings themselves, and sends the corrupted parties their shares.
      iv. In either case, $\mathcal{S}$ computes the sharing

$$[\boldsymbol{\gamma} * \boldsymbol{\theta}]_{n-1} \leftarrow [\boldsymbol{\gamma}]_{n-k} * \overline{[(\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1) * (\boldsymbol{b}_2 - \boldsymbol{b}_1)]}_{k-1} \\ + [\boldsymbol{\gamma} * \boldsymbol{b}_1]_{n-k} * \overline{[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]}_{k-1} + \rho \cdot [\boldsymbol{\gamma} * \boldsymbol{a}_1]_{n-k} * \overline{[\boldsymbol{b}_2 - \boldsymbol{b}_1]}_{k-1} + \\ \rho \cdot [\boldsymbol{\gamma} * \boldsymbol{c}_1]_{n-k} - [\boldsymbol{\gamma} * \boldsymbol{c}_2]_{n-k}.$$

   (e) To simulate $\pi_{\mathsf{check\text{-}zero}}$:
      i. $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{coin}}$ by sampling random $\chi_1, \ldots, \chi_M$ and sending them to the adversary.
      ii. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary the corrupted parties' shares of $[\mathbf{0}]_{n-1}$, and then sampling the rest of the sharing randomly, based on the corrupted parties' shares and $\mathbf{0}$.
      iii. Then $\mathcal{S}$ computes $[\boldsymbol{\theta}]_{n-1} \leftarrow [\mathbf{0}]_{n-1} + \sum_{l=1}^{M} \chi_l \cdot [\boldsymbol{\theta}_l]_{n-1}$.
      iv. Next $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{commit}}$ by receiving from the adversary the corrupted parties' shares of $[\boldsymbol{\theta}]_{n-1}$.
      v. Finally, $\mathcal{S}$ sends the adversary the honest parties' shares of $[\boldsymbol{\theta}]_{n-1}$ and receives the corrupted parties' shares from the adversary. If the corrupted parties' shares are different from the committed values or the computed shared value $\boldsymbol{\theta} \neq \mathbf{0}$, then $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$, $\mathsf{abort}$.
   (f) Finally, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{verify\text{-}deg}}$ by sending to the adversary the corrupted parties' shares of each $[\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1]_{k-1}$, $[\boldsymbol{b}_2 - \boldsymbol{b}_1]_{k-1}$, and $[(\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1) * (\boldsymbol{b}_2 - \boldsymbol{b}_1)]_{k-1}$, from $\pi_{\mathsf{sacrifice}}$, then sending $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ if $\mathsf{bad\text{-}shr} = 1$ or the adversary sends $\mathcal{S}$ $\mathsf{abort}$.

(g) Additionally, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary the corrupted parties' shares of $[\mathbf{0}_1]_{n-1}$ and $[\mathbf{0}_2]_{n-1}$, and then sampling the rest of the sharings randomly, based on the corrupted parties' shares and $\mathbf{0}$.

$\mathcal{S}$ then sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ the corrupted parties' shares of each $[\![\boldsymbol{a}_l]\!]_{n-k}$, $[\![\boldsymbol{b}_l]\!]_{n-k}$, $[\![\boldsymbol{c}_l]\!]_{n-k}$, as well as $[\mathbf{0}_1]_{n-1}$ and $[\mathbf{0}_2]_{n-1}$.

4. **Preparing Random Sharings for the Computation Verification**:
   (a) $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary the corrupted parties' shares of $[\mathbf{0}_1]_{n-1}$ and $[\mathbf{0}_2]_{n-1}$, and then sampling the rest of the sharings randomly, based on the corrupted parties' shares and $\mathbf{0}$. Note that if the adversary later corrupts more parties, $\mathcal{S}$ can just send their shares to the adversary.
   (b) Then, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{auth\text{-}rand}}$ by receiving from the adversary the corrupted parties' shares of $[\boldsymbol{\gamma}]_{n-k}$ and then either $\mathsf{abort}$ or the corrupted parties' shares of $[\![\boldsymbol{r}]\!]_{n-k}$. In the former case, $\mathcal{S}$ sends $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$; otherwise, it samples random sharing $[\![\boldsymbol{r}]\!]_{n-k}$ consistent with the corrupted parties' shares. Note that if the adversary later corrupts more parties, $\mathcal{S}$ can just send their shares to the adversary.
   (c) $\mathcal{S}$ then emulates $\mathcal{F}_{\mathsf{rand}}$ by receiving from the adversary the corrupted parties' shares of $\langle 0_1 \rangle$ and $\langle 0_2 \rangle$, and then sampling the rest of the sharings randomly, based on the corrupted parties' shares and 0.

   $\mathcal{S}$ then sends to $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ the corrupted parties' shares of $[\mathbf{0}_1]_{n-1}$ and $[\mathbf{0}_2]_{n-1}$, as well as $(\langle r \rangle, \langle \gamma \cdot r \rangle)$.

Now we argue that the real and ideal worlds are indistinguishable. For sampling of the MAC key, it is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{rand}}$ as in the real world. Also, the shares of $[\boldsymbol{\gamma}]_{n-k}$ that the honest parties output in the real world are distributed randomly given the corrupted parties' shares, by definition of $\mathcal{F}_{\mathsf{rand}}$. This is exactly the same as how $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples the honest parties' shares.

For the sharings of input and output gates: It is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{triple}}$ as in the real world. From above, we can also argue that $\mathcal{S}$'s simulation of $\pi_{\mathsf{auth}}$ is as in the real world. It is also clear that $\mathcal{S}$'s emulation of $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ and $\mathcal{F}_{\mathsf{rand}}$ are identical to the real world. Now we just need to argue that the shares that the honest parties output in the real world are identical to that of the ideal world. By definition of $\mathcal{F}_{\mathsf{triple}}$ and $\mathcal{F}_{\mathsf{rand}}$, the shares of $[\boldsymbol{r}_l]_{n-k}$, $[\boldsymbol{\Delta}_l]_{n-k}$, and $[\mathbf{0}]_{n-1}$ that the honest parties output in the real world are distributed randomly given the corrupted parties' shares (and $\mathbf{0}$ for $[\mathbf{0}]_{n-1}$). This holds even for $[\boldsymbol{r}_l]_{n-k}$, which is masked by random $[\boldsymbol{r}'_l]_{n-k}$ in $\pi_{\mathsf{auth}}$. This is exactly the same as how $\mathcal{F}_{\mathsf{prep\text{-}mal}}$ samples the honest parties' shares. In both the real and ideal worlds, $[\boldsymbol{\gamma} * \boldsymbol{r}_l]_{n-k}$ is distributed randomly given the corrupt parties' shares together with $\boldsymbol{\gamma} * \boldsymbol{r}_l$. The same can be said about $[\boldsymbol{\Delta}_l * \boldsymbol{r}_l]_{n-k}$, where in the real world, the corrupted parties shares are also changed so that the secret is $\boldsymbol{\Delta}_l * \boldsymbol{r}_l$.

For the packed, authenticated beaver triples: It is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{triple}}$ and $\mathcal{F}_{\mathsf{deg\text{-}reduce}}$ as in the real world. From above, we can also argue that $\mathcal{S}$'s simulation of $\pi_{\mathsf{auth}}$ is as in the real world. In the simulation of $\pi_{\mathsf{sacrifice}}$, it is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{coin}}$ as in the real world. It is also clear that the opening and redistribution of $[\overline{\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1}]_{k-1}$, $[\overline{\boldsymbol{b}_2 - \boldsymbol{b}_1}]_{k-1}$, and $[\overline{(\boldsymbol{a}_2 - \rho \cdot \boldsymbol{a}_1) * (\boldsymbol{b}_2 - \boldsymbol{b}_1)}]_{k-1}$

is as in the real world. In the simulation of $\pi_{\textsf{check-zero}}$ it is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\textsf{coin}}$, $\mathcal{F}_{\textsf{rand}}$, and $\mathcal{F}_{\textsf{commit}}$ as in the real world. It is also clear that it computes $[\boldsymbol{\theta}]_{n-1}$ as in the real world. So, the honest parties abort in the ideal world if and only if they do in the real world. Finally, it is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\textsf{verify-deg}}$ and $\mathcal{F}_{\textsf{rand}}$ as in the real world. Now, from Lemma 14, we know that if any $\boldsymbol{\gamma} * \boldsymbol{\theta}_{l^*}$ is nonzero, and random and unknown to the adversary, then the honest parties will abort with all-but-negligible probability. In particular, this means that each $[\boldsymbol{c}_1]_{n-k}$ cannot have any error, for otherwise, the corresponding $\boldsymbol{\gamma} * \boldsymbol{\theta}_{l^*}$ would contain $\rho \cdot \boldsymbol{\gamma} * \boldsymbol{\delta}_1$ and thus be random and unknown to the adversary. Observe also that in both worlds, if the adversary does not introduce any errors, then $\boldsymbol{\gamma} * \boldsymbol{\theta} = \mathbf{0}$. Now we just need to argue that the shares that the honest parties output in the real world are identical to that of the ideal world. In both worlds, it is clear that $[\boldsymbol{a}_1]_{n-k}, [\boldsymbol{b}_1]_{n-k}, [\mathbf{0}_1]_{n-1}, [\mathbf{0}_2]_{n-1}$ are randomly distributed given the corrupted parties' shares (and also $\mathbf{0}$ for the last two). Each $[\boldsymbol{c}_l]_{n-k}$ is distributed randomly given the corrupted parties' shares and $\boldsymbol{a}_l * \boldsymbol{b}_l$. The same can be said of $[\boldsymbol{\gamma} * \boldsymbol{a}_l]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{b}_l]_{n-k}, [\boldsymbol{\gamma} * \boldsymbol{c}_l]_{n-k}$, given the corrupted parties' shares and $\boldsymbol{\gamma} * \boldsymbol{a}_l$, $\boldsymbol{\gamma} * \boldsymbol{b}_l$, and $\boldsymbol{\gamma} * \boldsymbol{c}_l$, respectively.

For the random sharings for computation verification: it is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\textsf{rand}}$ and $\mathcal{F}_{\textsf{auth-rand}}$ as in the real world. It is also clear that in both worlds, $[\mathbf{0}_1]_{n-1}, [\mathbf{0}_2]_{n-1}$ are distributed randomly given the corrupted parties' shares and $\mathbf{0}$. Also, the honest parties' shares of $\langle r \rangle, \langle \gamma \cdot r \rangle$ are distributed randomly.

Thus, we have argued that the real and ideal worlds are indistinguishable with all-but-negligible probability. $\square$