Highly Efficient Server-Aided Multiparty Subfield VOLE Distribution Protocol

Dongyu Wu

Beijing Institute of Mathematical Sciences and Applications
wudongyu@bimsa.cn

Abstract. In recent development of secure multi-party computation (MPC), pseudorandom correlations of subfield vector oblivious linear evaluation (sVOLE) type become popular due to their amazing applicability in multi-dimensional MPC protocols such as privacy-preserving biometric identification and privacy-preserving machine learning protocols. In this paper, we introduce a novel way of sVOLE distribution in three-party and four-party honest majority settings with the aid of a trusted server. This new method significantly decreases the communication cost and the memory storage of random sVOLE instances. On the other hand, it also enables a streamline distribution process that can generate a sVOLE instance of an arbitrary length, which results in 100 percent of utility rate of random sVOLE in multi-dimensional MPC protocols while preserving complete precomputability.

Keywords: MPC· server-aided MPC· tensor triple· subfield VOLE

1 Introduction

1.1 Motivation

Secure multiparty computation (MPC) is one of the central subfields in cryptography. MPC aims to accomplish a joint evaluation of a function over inputs provided by multiple parties without revealing any extra information. Due to its feature on protection of the inputs, it has been widely used in analysis and processing of private or sensitive data held by multiple parties.

Meanwhile, apart from the classical correlation generation protocols such as oblivious transfer (OT) and Beaver triples, researchers have also attempted to find new kinds of correlations (for example [1]) that can fulfill different needs of MPC protocols and perform more efficiently compared to classical ways [2–5]. Among these variants, subfield vetor oblivious linear evaluation (sVOLE), as an efficient way to provide correlated random vector sharings for two parties, has been widely used as a convenient auxiliary tool to accomplish circuit-based MPC involving multi-dimensional inputs.

In [6], the authors explicitly described the connection between sVOLE and multi-dimensional secure two-party computation through the notion of tensor triple. The tensor triple technique introduced in the article not only enables the secure computation of matrix multipliction through a direct processing of vectors instead of breaking the computation entrywise, but preserving the essential property of the precomputability of multiplicative triples, namely the participants do not need any prior knowledge of the dimensions of the matrices involved in further protocols to generate tensor triples.

As multi-dimensional MPC is widely and heavily used not only in privacy-preserving biometric authentication and identification, but also various privacy-preserving machine learning protocols, it is natural to expect an efficient way to fulfill secure multi-dimensional computation in 3PC or even generic MPC cases, as the number of data providers in machine learning is not necessarily restricted to two. Furthermore, although there exists multiple transformation methods from a generic 2PC simple bilinear correlation to its MPC generalization (for instance [7]), the efficiency is often compromised and the transformation usually does not provide any optimization. Therefore, to fulfill the need of efficient privacy-preserving machine learning protocols, it is required an optimized and well-designed tensor triple generation process has to be established.

Server-aided secure multi-party computation was first introduced by Beaver in [8]. The existence of a trusted server as explained can help reduce the communication and computation costs of the MPC protocols considerably. There also have been researches (for instance [9, 10]) which fulfilled the usual Beaver correlation and generic correlation distributions with the aid of a trusted server and achieved an apparent better performance in efficiency. Hence an exploration of a specialized server-aided distribution of the sVOLE and tensor triple correlation is desirable. It is expected the participation of a trusted server can significantly reduce the communication, computation, and even memory storage costs of the distribution protocol.

1.2 Our contribution

The goal of this paper is to explore a more efficient way to fulfill multi-dimensional secure multi-party computation in the case where the number of participants is larger than two. In this paper, we generalize canonically the notion of tensor triples to N parties. We also invented a streamlined server-aided tensor triple generation protocol in the honest majority setting, in which the server can continuously provide tensor triples to all parties in an amazingly fast speed and the storage space of triples for each participant is also largely optimized to obtain a linear overhead, through a novel way by breaking the rank 1 matrix in the third entry of a tensor triple into low rank matrices instead of full rank ones. Furthermore, in such a distribution protocol, the utility rate of the triples can achieve up to 100%. Hence our server-aided tensor triple distribution protocol provides an optimal utilization efficiency while achieving the full precomputability.

1.3 Related works

A generic way to transform a 2PC sVOLE protocol to a generic MPC sVOLE protocol has been introduced in [7]. Although not defined identically, the following works [11, 12] also provide a generation of sVOLE protocol in a generic MPC situation and give a corresponding design for implementation. It should be mentioned that the original definition of VOLE does not generalize in a canonical way. We will explain the difference in these generalization in detail in further sections, but it should be emphasized our work proposed a novel and very efficient way for tensor triple generation and distribution, and its advantages will also be explained in detail.

The idea of distributing multiplicative triples with the aid from a trusted server (or equivalently referred to as Triple-as-a-Service) has been explored by many researchers (See [9, 10]), whilst the main goal of such a protocol is to transfer offline burdens to the server. In our research, however, this is not the only reason of importing the server. The server also enables a streamline distribution of tensor triples and the communication cost can be shrinked with a specialized design in the server-aided case. Therefore, the existence of the server grants significant benefits for the tensor triple distribution and may well be an optimized way to generate them.

1.4 Arrangement of paper

In Chapter 2, we introduce define necessary notions and primitives in MPC. In Chapter 3, we recall the notion of tensor triple in two-party case and briefly explain the generalization and provide a brief comparison with different definitions in literature. In Chapter 4, we explicitly describe the server-aided tensor triple distribution protocols in both 3PC and 4PC cases in detail, and discuss their corresponding theoretical efficiency and security. In Chapter 5 we present implementations of protocols as well as results of the experiments after the execution are presented.

1.5 Security model

We consider the security in the universal composability (UC) model. All protocols involved in the rest of the paper are secure against semi-honest and computationally bounded adversaries. We further impose the honest majority assumption, namely the adversary cannot collude with more than or equal to half of the participants. In the case where a server is present, the server is assumed to be a trusted one.

2 Preliminaries and notations

2.1 Oblivious transfer (OT)

We provide a very brief introduction of oblivious transfer together with its multiple invariants in order to import all possible notations to be used. In an oblivious trnasfer [13], the sender with a pair of messages (m_0, m_1) interacts with the receiver with a choice bit b. The result ensures that the receiver learns m_b but obtains no knowledge of m_{1-b}, while the sender obtains no knowledge of b. In an OT extension protocol OT_l^n, the input of the sender is n message pairs $(m_{i,0}, m_{i,1}) \in \{0,1\}^{2l}$ and the input of the receiver is a string $\mathbf{b} \in \{0,1\}^n$. The result allows the receiver to learn $m_{i,\mathbf{b}[i]}$ for $1 \le i \le n$. In a Random OT (ROT), the sender inputs nothing beforehand but obtains two random strings in the outputs as the message pair, and the receiver inputs nothing either but obtains the choice bit together with the selected message afterwards. Similiarly, a batched version of ROT (or also known as OT extension, see [14–16]) which generates n message pairs of bit-length l is denoted by ROT_l^n. A correlated OT (COT) [5, 17, 18] is a variant of ROT that allows the sender to predetermine a string Δ and obtain two correlated random strings as the message pair with their XOR equal Δ. The extension of COT denoted by COT_m^n allows the sender to choose $\Delta \in \mathbb{F}_{2^m}$. The protocol eventually provides two uniformly distributed vectors $\mathbf{u} \in \mathbb{F}_2^n, \mathbf{v} \in \mathbb{F}_{2^m}^n$ to the receiver, and $\mathbf{v} \oplus (\mathbf{u} \cdot \Delta)$ to the sender.

2.2 Subfield vector oblivious linear evaluation (sVOLE)

A subfield vector oblivious linear evaluation protocol is a generalization of COT_m^n to an arbitrary finite base field. A vector oblivious linear evaluation (VOLE) allows one party to obtain two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^n$, and the other party a scalar $x \in \mathbb{F}_p$ and the linear evaluation $\mathbf{u}x + \mathbf{v}$.

In further chapters, we will be mainly using the random variants of COT_m^n and sVOLE functionalities defined in Fig. 1. It is not difficult to see that RCOT_m^n is a special case of $\mathscr{F}_{\mathrm{RsVOLE}}^{p,m,n}$ when $p = 2$.

Functionality RCOT^n_m	Functionality $\mathscr{F}^{p,m,n}_{\mathrm{RsVOLE}}$
Players: The sender S and the receiver R.	**Players:** The sender S and the receiver R.
Inputs: None.	**Inputs:** A dimension pair (n, m).
Outputs:	**Outputs:**
– S outputs $\mathbf{v} \in \mathbb{F}^n_{2^m}, \Delta \in \mathbb{F}_{2^m}$;	– S outputs $x \in \mathbb{F}_{p^m}, \mathbf{v} \in \mathbb{F}^n_{p^m}$;
– R outputs $\mathbf{u} \in \mathbb{F}^n_2, \mathbf{v} \oplus (\mathbf{u} \cdot \Delta) \in \mathbb{F}^n_{2^m}$	– R outputs $\mathbf{u} \in \mathbb{F}^n_p, x\mathbf{u} + \mathbf{v} \in \mathbb{F}^n_{p^m}$.

Fig. 1. Functionalities of Random COT and Random sVOLE

VOLE protocols with semi-honest and computational security in OT-hybrid model have been defined in [3, 5]. Subfield VOLE, as an important variant of VOLE, has also been studied and implemented in various ways (See [4, 5, 18–21]).

3 Tensor triple

Tensor triple defined in [6] is an efficient way to offer multi-dimensional masks to vectors for secure tensor computations and matrix products. This type of randomization is very useful in privacy-preserving batched biometric idenfication and privacy-preserving machine learning protocols which usually involves numerous matrix products of matrices with large dimensions.

3.1 Generic definition of tensor triple

We will basically follow the definition described in [6].

Definition 1 (Tensor triple)**.** *By definition, a tensor triple* $(\boldsymbol{u}, \boldsymbol{v}, W)$ *consists of data* $\boldsymbol{u} \in K^m, \boldsymbol{v} \in K^n, W \in M_{m\times n}(K)$ *satisfying* $\boldsymbol{u} \otimes \boldsymbol{v} = \boldsymbol{u}\boldsymbol{v}^T = W$.

Definition 2 (Tensor triple sharing)**.** *Let* $P_1, ..., P_N$ *be the participants of a secure multi-party computation protocol over an ambient finite field or ring* K*. By definition, a tensor triple sharing scheme provides two vectors and one matrix*

$$\boldsymbol{u}_i \in K^m, \quad \boldsymbol{v}_i \in K^n, \quad W_i \in M_{m\times n}(K)$$

for the participant P_i*, such that* $\boldsymbol{u}_i, \boldsymbol{v}_i$ *form secret sharings of* $\boldsymbol{u} \in K^m, \boldsymbol{v} \in K^n$ *respectively, with the relation* $\boldsymbol{u} \otimes \boldsymbol{v} = \boldsymbol{u}\boldsymbol{v}^T = \sum_{i=1}^N W_i$*. Shares of* $\boldsymbol{u}, \boldsymbol{v}$ *and* W *will be denoted by* $[\boldsymbol{u}], [\boldsymbol{v}]$ *and* $[W]$ *if indices are not particulary involved. For our convenience, such a triple will be called an* (m, n)*-triple.*

Tensor triple can be regarded as a generalization of the Beaver triple when $m = n = 1$. Therefore we note if a distribution protocol of tensor triples is independent of the choice of the dimensions, it can also fulfill the need of a distribution of the usual Beaver triples, and therefore serving as the pre-compuation procedure for a generic GMW or BGW protocol.

Secure Multi-Party Outer Product $\mathsf{Out}(\mathbf{a}, \mathbf{b})$

Input: P_i inputs shares $[\mathbf{a}]_i, [\mathbf{b}]_i$ of two vectors $\mathbf{a} \in K^m, \mathbf{b} \in K^n$.
Primitive: A tensor triple generation protocol $\mathsf{TT.Gen}(m, n, \lambda)$
Pre-processing Phase: The participants perform a $\mathsf{TT.Gen}(m, n, \lambda)$ protocol. P_i outputs $[\mathbf{u}]_i \in K^m, [\mathbf{v}]_i \in K^n, [W]_i \in M_{m\times n}(K)$.
Initial Phase: P_i computes $[\mathbf{s}]_i \leftarrow [\mathbf{a}]_i - [\mathbf{u}]_i$ and $[\mathbf{t}]_i \leftarrow [\mathbf{b}]_i - [\mathbf{v}]_i$.
Interacting Phase:

1. P_i annouces publicly to all parties $[\mathbf{s}]_i$ and $[\mathbf{t}]_i$;
2. All parties recover $\mathbf{s}$ and $\mathbf{t}$ from the annoucement;
3. P_1 secretly computes $[\mathbf{a} \otimes \mathbf{b}]_1 \leftarrow \mathbf{s} \otimes \mathbf{t} + [\mathbf{u}]_1 \otimes \mathbf{t} + \mathbf{s} \otimes [\mathbf{v}]_1 + [W]_1$, and each other party P_i for $i \neq 1$ secretly computes $[\mathbf{a} \otimes \mathbf{b}]_i \leftarrow [\mathbf{u}]_i \otimes \mathbf{t} + \mathbf{s} \otimes [\mathbf{v}]_i + [W]_i$.

Outputs: P_i outputs $[\mathbf{a} \otimes \mathbf{b}]_i$.

Fig. 2. Tensor-triple-based secure outer product protocol

We also recall in Fig. 2 the secure generic multi-party outer product protocol introduced in [6]. This also induces a secure matrix product protocol when applying the outer product expansion formula $AB = \sum \mathbf{a}_i \otimes \mathbf{b}_i$.

3.2 Comparison with other definitions in literature

In this section we would like to introduce other definitions of 3PC or MPC sVOLE. The generalization to three-party or generic multi-party cases can lead to several possible ways to fit the need of different MPC protocols. In this section we apply the conventional terminology that $\mathbb{K} \subset \mathbb{F}$ is a fixed field extension of degree m.

Boyle et al. In [7], the generalization of two-party sVOLE is described in the notion of the so-called Simple Bilinear Correlation. The functionality provides each of the participants $P_1, ..., P_N$ with $(\mathbf{u}_i, x_i, \mathbf{v}_i) \in \mathbb{K}^n \times \mathbb{F} \times \mathbb{F}^n$, where $\sum x_i \sum \mathbf{u}_i = \sum \mathbf{v}_i$. This type of generalization coincides with our definition of tensor triples up to a canonical isomorphism of linear spaces $\phi : \mathbb{F} \to \mathbb{K}^m$.

Qiu, Yang, Yu and Zhou In [11], the functionality of multiparty sVOLE provides one main participant P_0 with $(\mathbf{u}_i, \mathbf{w}_i) \in \mathbb{K}^n \times \mathbb{F}^n$ for all $i = 1, ..., N$, and each of the participants $P_1, ..., P_N$ with $(\mathbf{v}_i, \Delta_i) \in \mathbb{F}^n \times \mathbb{F}$, such that $\mathbf{w}_i = \mathbf{v}_i + \Delta_i \mathbf{u}_i$ for all $i = 1, ..., N$. This functionality is utilized to fulfill the need of an efficient maliciously secure multiparty PSI.

Zhang In [12], the functionality of the so-called pseudorandom MP-sVOLE generator is described as such to provide P_1 with $(\mathbf{a}_0, \mathbf{a}_1) \in \mathbb{F}^n \times \mathbb{K}^n$, P_N with $(\mathbf{a}_N, x_1, ..., x_{N-1}) \in \mathbb{F}^n \times \mathbb{F}^{N-1}$, and P_i with $\mathbf{a}_i \in \mathbb{K}^n$, such that

$$\mathbf{a}_0 = x_1\mathbf{a}_1 + ... + x_{N-1}\mathbf{a}_{N-1} + \mathbf{a}_N.$$

The functionality is used to generate a multiparty PSI protocol with a comparatively lower communication cost.

In this paper, our notion of tensor triple generates canonically to multiparty cases, and we emphasize this generalization coincides with the notion of multiparty simple bilinear correlation defined in [7] up to a canonical isomorphism of vector spaces.

4 Server-aided tensor triple generation

A third-party with computational power may be eligible to provide triples for multiple parties in a much more efficient way. This idea has been explored by many people, such as [9, 10]. The protocol described in all these papers can be used almost directly to generate tensor triples for multiple parties (not necessarily only two). The flexibility of tensor triple allows the server to provide triples of fixed dimensions while fulfilling the needs for all lower dimensional computations. More specifically, the dimensions of the triples may be predetermined, as a generic (n, n)-triple could be tailored to serve as a pair of triples of dimensions (s, t) for any $s, t < n$. This means the parties need not know the precise dimensions in advance for the preprocessing procedure. A great advantage is that a specialized server may serve as the triple generator for multiple sets of multiple parties in order to speed up all preprocessing procedure. Furthermore, we will explain later that our generation protocol can technically remove the restriction of an upper bound n.

4.1 Functionality

Functionality $\mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}$

The functionality interacts with three parties P_1, P_2, P_3 and an adversary $\mathcal{A}$ who may corrupt with exactly one of the three parties. K is a finite field and m, n parametrize the dimensions.

1. Upon receiving (init, sid_i) from all three parties P_1, P_2, P_3, sample uniformly random vectors
$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{a}_1, \mathbf{b}_1, \mathbf{b}_2, \mathbf{a}_3, \mathbf{b}_3 \leftarrow_\$ K^m,$$
$$\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{c}_1, \mathbf{d}_1, \mathbf{d}_2, \mathbf{c}_3, \mathbf{d}_3 \leftarrow_\$ K^n,$$
such that $\mathbf{b}_1 = \mathbf{b}_3$, $\mathbf{d}_1 = \mathbf{d}_3$, $\mathbf{b}_2 - \mathbf{b}_1 = \mathbf{a}_1 - \mathbf{a}_3 = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3$, $\mathbf{d}_2 - \mathbf{d}_1 = \mathbf{c}_1 - \mathbf{c}_3 = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3$. If P_i is corrupted, receive $(\mathbf{x}_i, \mathbf{y}_i)$ from $\mathcal{A}$ and set $\mathbf{b}_2 = \mathbf{b}_1 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3$, $\mathbf{a}_3 = \mathbf{a}_1 - \mathbf{x}_1 - \mathbf{x}_2 - \mathbf{x}_3$, $\mathbf{d}_2 = \mathbf{d}_1 + \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3$ and $\mathbf{c}_3 = \mathbf{c}_1 - \mathbf{y}_1 - \mathbf{y}_2 - \mathbf{y}_3$.
2. Send:
$(\mathbf{x}_1, \mathbf{y}_1, \mathbf{a}_1, \mathbf{b}_1, \mathbf{c}_1, \mathbf{d}_1)$ to P_1,
$(\mathbf{x}_2, \mathbf{y}_2, \mathbf{b}_2, \mathbf{d}_2)$ to P_2 and
$(\mathbf{x}_3, \mathbf{y}_3, \mathbf{a}_3, \mathbf{b}_3, \mathbf{c}_3, \mathbf{d}_3)$ to P_3.

Fig. 3. Functionality of 3PC tensor triple distribution

Before we proceed introducing the distribution protocol, we first give the definition of the functionality we attempt to fulfill in Fig. 3. The main difference of this new functionality from the classical sVOLE functionality is a lower rank matrix in the third component. This enables an extremely faster communication in a high dimensional parameter setting and also allows us to formulate a streamline distribution protocol which we will explain in detail in further sections.

We first define the functionality $\mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}$ for 3PC tensor triple generation. As shown in the output, the three participants may store the share of W separately in vectors without computing out the full matrix and only expand them as $W_1 = -\mathbf{a}_1 \otimes \mathbf{d}_1 - \mathbf{b}_1 \otimes \mathbf{c}_1 - \mathbf{b}_1 \otimes \mathbf{d}_1$, $W_2 = \mathbf{b}_2 \otimes \mathbf{d}_2$ and $W_3 = \mathbf{a}_3 \otimes \mathbf{d}_3 + \mathbf{b}_3 \otimes \mathbf{c}_3$ when it is necessary (for instance to compute the final result). The following computation shows the correctness of the protocol:

$$\begin{aligned}
&W_1 + W_2 + W_3 \\
=& -\mathbf{a}_1 \otimes \mathbf{d}_1 - \mathbf{b}_1 \otimes \mathbf{c}_1 - \mathbf{b}_1 \otimes \mathbf{d}_1 + \mathbf{b}_2 \otimes \mathbf{d}_2 + \mathbf{a}_3 \otimes \mathbf{b}_3 + \mathbf{c}_3 \otimes \mathbf{d}_3 \\
=& -\mathbf{a}_1 \otimes \mathbf{d}_1 - \mathbf{b}_1 \otimes \mathbf{c}_1 - \mathbf{b}_1 \otimes \mathbf{d}_1 + (\mathbf{b}_1 + \mathbf{x}) \otimes (\mathbf{d}_1 + \mathbf{y}) \\
&+ (\mathbf{a}_1 - \mathbf{x}) \otimes \mathbf{d}_1 + \mathbf{b}_1 \otimes (\mathbf{c}_1 - \mathbf{y}) \\
=&\mathbf{x} \otimes \mathbf{y}
\end{aligned}$$

To show that the functionality does provide a sufficient masking. We need the proposition below.

Theorem 3. *For any adversary $\mathcal{A}$ we have*

$$Pr\left[\begin{array}{c} (\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}'_1, \boldsymbol{y}'_1) \leftarrow \mathcal{A}(1^\lambda) \\ (\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \\ \boldsymbol{b}_2, \boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{d}_2, \boldsymbol{c}_3, \boldsymbol{d}_3) \\ \leftarrow \mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}(\boldsymbol{x}_1, \boldsymbol{y}_1, \mathsf{Corrupt} = 1) \\ : \mathcal{A}(\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1, \boldsymbol{d}_1, \\ \boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{x}_3, \boldsymbol{y}_3) = 1 \end{array}\right] = Pr\left[\begin{array}{c} (\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}'_1, \boldsymbol{y}'_1) \leftarrow \mathcal{A}(1^\lambda) \\ (\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \\ \boldsymbol{b}_2, \boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{d}_2, \boldsymbol{c}_3, \boldsymbol{d}_3) \\ \leftarrow \mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}(\boldsymbol{x}'_1, \boldsymbol{y}'_1, \mathsf{Corrupt} = 1) \\ : \mathcal{A}(\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1, \boldsymbol{d}_1, \\ \boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{x}_3, \boldsymbol{y}_3) = 1 \end{array}\right]$$

Similarly, we have

$$Pr\left[\begin{array}{c} (\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{x}'_2, \boldsymbol{y}'_2) \leftarrow \mathcal{A}(1^\lambda) \\ (\boldsymbol{b}_2, \boldsymbol{d}_2, \boldsymbol{x}_1, \boldsymbol{x}_3, \boldsymbol{a}_1, \boldsymbol{b}_1, \\ \boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{y}_1, \boldsymbol{y}_3, \boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{c}_3, \boldsymbol{d}_3) \\ \leftarrow \mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}(\boldsymbol{x}_2, \boldsymbol{y}_2, \mathsf{Corrupt} = 2) \\ : \mathcal{A}(\boldsymbol{b}_2, \boldsymbol{d}_2, \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_3, \boldsymbol{y}_3) = 1 \end{array}\right] = Pr\left[\begin{array}{c} (\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{x}'_2, \boldsymbol{y}'_2) \leftarrow \mathcal{A}(1^\lambda) \\ (\boldsymbol{b}_2, \boldsymbol{d}_2, \boldsymbol{x}_1, \boldsymbol{x}_3, \boldsymbol{a}_1, \boldsymbol{b}_1, \\ \boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{y}_1, \boldsymbol{y}_3, \boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{c}_3, \boldsymbol{d}_3) \\ \leftarrow \mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}(\boldsymbol{x}'_2, \boldsymbol{y}'_2, \mathsf{Corrupt} = 2) \\ : \mathcal{A}(\boldsymbol{b}_2, \boldsymbol{d}_2, \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_3, \boldsymbol{y}_3) = 1 \end{array}\right]$$

and

$$Pr\left[\begin{array}{c} (\boldsymbol{x}_3, \boldsymbol{y}_3, \boldsymbol{x}'_3, \boldsymbol{y}'_3) \leftarrow \mathcal{A}(1^\lambda) \\ (\boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{c}_3, \boldsymbol{d}_3, \boldsymbol{x}_1, \boldsymbol{x}_2, \\ \boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{d}_2) \\ \leftarrow \mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}(\boldsymbol{x}_3, \boldsymbol{y}_3, \mathsf{Corrupt} = 3) \\ : \mathcal{A}(\boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{c}_3, \boldsymbol{d}_3, \\ \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2) = 1 \end{array}\right] = Pr\left[\begin{array}{c} (\boldsymbol{x}_3, \boldsymbol{y}_3, \boldsymbol{x}'_3, \boldsymbol{y}'_3) \leftarrow \mathcal{A}(1^\lambda) \\ (\boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{c}_3, \boldsymbol{d}_3, \boldsymbol{x}_1, \boldsymbol{x}_2, \\ \boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{d}_2) \\ \leftarrow \mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}(\boldsymbol{x}'_3, \boldsymbol{y}'_3, \mathsf{Corrupt} = 3) \\ : \mathcal{A}(\boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{c}_3, \boldsymbol{d}_3, \\ \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2) = 1 \end{array}\right]$$

Well-behaved as the distribution protocol seems, it does not ensure the output masking. The key reason the distribution protocol does not provide the masking is because after all W_i is a low rank matrix which cannot provide a sufficient mask to anything to be announced. But we also need to emphasize that W_i is not supposed to be published directly in our setting. To resolve such a problem, the participants may for instance apply a pseudorandom zero-sharing (PRZS) protocol introduced in [22] or any similar secret sharing scheme to mask the final share before the publication. This process is only needed in the final step and therefore will not increase the online communicational and computational costs by much.

Alternatively, if such an extra procedure is not preferred, in applications it is suggested the MPC function to be computed need to amalgamate multiple low rank matrices in the third component of different tensor triples to ensure the mask is enough. For instance, when using tensor triples for secure matrix product protocol, we suggest the number of columns of the first matrix, which is also the number of rows of the second matrix, to be at least half of the maximal number of rows of the first matrix and columns of the second matrix. In our implementation, however, we will take the example of square matrix multiplication of size 1024×1024 and hence does not have any such problem. The output security is dependent of the MPC protocol to be processed and cannot be formally stated generally. We here only state explicitly the output security for the secure square matrix product protocol below for reference. The proofs of the two security statements will be provided in the appendix.

Theorem 4. *For any adversary $\mathcal{A}$ we have*

$$Pr\left[\begin{array}{c}(\{\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l},\boldsymbol{x}'_{1,l},\boldsymbol{y}'_{1,l}\})\leftarrow\mathcal{A}(1^\lambda),\\(\{\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\boldsymbol{x}_{2,l},\boldsymbol{x}_{3,l},\\\boldsymbol{b}_{2,l},\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{y}_{2,l},\boldsymbol{y}_{3,l},\boldsymbol{d}_{2,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l}\})\\\leftarrow\mathscr{F}^{K,n,n}_{\mathsf{3PCTT}}(\{\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l}\},\mathsf{Corrupt}=1)\\:\mathcal{A}(\{\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\\\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l},\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l}\},W_2,W_3)=1\\(l=1,...,n)\end{array}\right]=Pr\left[\begin{array}{c}(\{\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l},\boldsymbol{x}'_{1,l},\boldsymbol{y}'_{1,l}\})\leftarrow\mathcal{A}(1^\lambda),\\(\{\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\boldsymbol{x}_{2,l},\boldsymbol{x}_{3,l},\\\boldsymbol{b}_{2,l},\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{y}_{2,l},\boldsymbol{y}_{3,l},\boldsymbol{d}_{2,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l}\})\\\leftarrow\mathscr{F}^{K,n,n}_{\mathsf{3PCTT}}(\{\boldsymbol{x}'_{1,l},\boldsymbol{y}'_{1,l}\},\mathsf{Corrupt}=1)\\:\mathcal{A}(\{\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\\\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l},\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l}\},W_2,W_3)=1\\(l=1,...,n)\end{array}\right]$$

computationally up to a negligible factor, where $W_i = \sum_{l=1}^{n} W_{i,l}$, $i = 1, 2, 3$. Similarly, we have

$$Pr\left[\begin{array}{c}(\{\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l},\boldsymbol{x}'_{2,l},\boldsymbol{y}'_{2,l}\})\leftarrow\mathcal{A}(1^\lambda)\\(\{\boldsymbol{b}_{2,l},\boldsymbol{d}_{2,l},\boldsymbol{x}_{1,l},\boldsymbol{x}_{3,l},\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\\\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{y}_{1,l},\boldsymbol{y}_{3,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l}\})\\\leftarrow\mathscr{F}^{K,n,n}_{\mathsf{3PCTT}}(\{\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l}\},\mathsf{Corrupt}=2)\\:\mathcal{A}(\{\boldsymbol{b}_{2,l},\boldsymbol{d}_{2,l},\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l},\\\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l}\},W_1,W_3)=1\\(l=1,...,n)\end{array}\right]=Pr\left[\begin{array}{c}(\{\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l},\boldsymbol{x}'_{2,l},\boldsymbol{y}'_{2,l}\})\leftarrow\mathcal{A}(1^\lambda)\\(\{\boldsymbol{b}_{2,l},\boldsymbol{d}_{2,l},\boldsymbol{x}_{1,l},\boldsymbol{x}_{3,l},\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\\\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{y}_{1,l},\boldsymbol{y}_{3,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l}\})\\\leftarrow\mathscr{F}^{K,n,n}_{\mathsf{3PCTT}}(\{\boldsymbol{x}'_{2,l},\boldsymbol{y}'_{2,l}\},\mathsf{Corrupt}=2)\\:\mathcal{A}(\{\boldsymbol{b}_{2,l},\boldsymbol{d}_{2,l},\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l},\\\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l}\},W_1,W_3)=1\\(l=1,...,n)\end{array}\right]$$

and

$$Pr\left[\begin{array}{c}(\{\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l},\boldsymbol{x}'_{3,l},\boldsymbol{y}'_{3,l}\})\leftarrow\mathcal{A}(1^\lambda)\\(\{\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l},\boldsymbol{x}_{1,l},\boldsymbol{x}_{2,l},\\\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\boldsymbol{b}_{2,l},\boldsymbol{y}_{1,l},\boldsymbol{y}_{2,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\boldsymbol{d}_{2,l}\})\\\leftarrow\mathscr{F}^{K,m,n}_{\mathsf{3PCTT}}(\{\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l}\},\mathsf{Corrupt}=3)\\:\mathcal{A}(\{\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l},\\\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l},\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l}\},W_1,W_2)=1\\(l=1,...,n)\end{array}\right]=Pr\left[\begin{array}{c}(\{\boldsymbol{x}_{3,l},\boldsymbol{y}_{3,l},\boldsymbol{x}'_{3,l},\boldsymbol{y}'_{3,l}\})\leftarrow\mathcal{A}(1^\lambda)\\(\{\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l},\boldsymbol{x}_{1,l},\boldsymbol{x}_{2,l},\\\boldsymbol{a}_{1,l},\boldsymbol{b}_{1,l},\boldsymbol{b}_{2,l},\boldsymbol{y}_{1,l},\boldsymbol{y}_{2,l},\boldsymbol{c}_{1,l},\boldsymbol{d}_{1,l},\boldsymbol{d}_{2,l}\})\\\leftarrow\mathscr{F}^{K,m,n}_{\mathsf{3PCTT}}(\{\boldsymbol{x}'_{3,l},\boldsymbol{y}'_{3,l}\},\mathsf{Corrupt}=3)\\:\mathcal{A}(\{\boldsymbol{a}_{3,l},\boldsymbol{b}_{3,l},\boldsymbol{c}_{3,l},\boldsymbol{d}_{3,l},\\\boldsymbol{x}_{1,l},\boldsymbol{y}_{1,l},\boldsymbol{x}_{2,l},\boldsymbol{y}_{2,l}\},W_1,W_2)=1\\(l=1,...,n)\end{array}\right]$$

4.2 Server-aided 3PC tensor triple distribution

3PC tensor triple distribution $\mathsf{TT.3PCDist}()$

Players: A server S. Parties P_1, P_2, P_3.
Distribution Phase:

1. Upon receiving (init, sid_i) from all three parties, S starts its distribution.
2. S randomly samples seeds $\mathsf{sd}_{\mathbf{x},1}, \mathsf{sd}_{\mathbf{x},2}, \mathsf{sd}_{\mathbf{x},3}, \mathsf{sd}_{\mathbf{y},1}, \mathsf{sd}_{\mathbf{y},2}, \mathsf{sd}_{\mathbf{y},3}$, $\mathsf{sd}_{\mathbf{x},\mathbf{a}}, \mathsf{sd}_{\mathbf{x},\mathbf{b}}, \mathsf{sd}_{\mathbf{y},\mathbf{c}}, \mathsf{sd}_{\mathbf{y},\mathbf{d}}$;
3. S sends $(\mathsf{sd}_{\mathbf{x},1}, \mathsf{sd}_{\mathbf{y},1}$, $\mathsf{sd}_{\mathbf{x},\mathbf{a}}, \mathsf{sd}_{\mathbf{x},\mathbf{b}}, \mathsf{sd}_{\mathbf{y},\mathbf{c}}, \mathsf{sd}_{\mathbf{y},\mathbf{d}})$ to P_1, $(\mathsf{sd}_{\mathbf{x},2}, \mathsf{sd}_{\mathbf{y},2})$ to P_2, and $(\mathsf{sd}_{\mathbf{x},3}, \mathsf{sd}_{\mathbf{y},3}, \mathsf{sd}_{\mathbf{x},\mathbf{b}}, \mathsf{sd}_{\mathbf{y},\mathbf{d}})$ to P_3.
4. S computes $\mathbf{x} = \sum_{i=1}^{3} \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},i})$, $\mathbf{y} = \sum_{i=1}^{3} \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},i})$, $\mathbf{a} = \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},\mathbf{a}})$, $\mathbf{b} = \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},\mathbf{b}})$, $\mathbf{c} = \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},\mathbf{c}})$, $\mathbf{d} = \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},\mathbf{d}})$. Then S computes block-by-block $\mathbf{u}_2 = \mathbf{b} + \mathbf{x}, \mathbf{v}_2 = \mathbf{d} + \mathbf{y}$, $\mathbf{u}_3 = \mathbf{a} - \mathbf{x}, \mathbf{v}_3 = \mathbf{c} - \mathbf{y}$ and continuously sends entries of $(\mathbf{u}_2, \mathbf{v}_2)$ to P_2 and entries of $(\mathbf{u}_3, \mathbf{v}_3)$ to P_3. Once S finishes the distribution of all $\mathbf{u}_2, \mathbf{v}_2, \mathbf{u}_3, \mathbf{v}_3$, it goes to Step 2 and repeats the process.
5. When S receives (pause, sid_i) from any party it pauses the distribution, and continues its computation and distribution upon receiving $(\mathsf{continue}, sid_i)$ from all three parties. When S receives (stop, sid_i) from all three parties, it stops the distribution and discards current data immediately.

Fig. 4. Server-aided 3PC tensor triple distribution protocol

3PC tensor triple utilization

For utilization in further protocols, P_1, P_2, P_3 need keep tracks of the indices $(\mathsf{flag}_{\mathbf{x}}, \mathsf{flag}_{\mathbf{y}})$ of the entries of $\mathbf{x}, \mathbf{y}$ they have already exhausted. When an (m, n)-tensor triple is needed,

- P_1 outputs $([\mathbf{x}]_1, [\mathbf{y}]_1, [W]_1)$, where

$$[\mathbf{x}]_1 = \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},1})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m], [\mathbf{y}]_1 = \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},1})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n],$$

  and $[W]_1 = -\mathsf{PRG}(\mathsf{sd}_{\mathbf{x},\mathbf{a}})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m] \otimes \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},\mathbf{d}})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n] - \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},\mathbf{b}})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m] \otimes \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},\mathbf{c}})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n] - \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},\mathbf{b}})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m] \otimes \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},\mathbf{d}})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n]$.
- P_2 outputs $([\mathbf{x}]_2, [\mathbf{y}]_2, [W]_2)$, where

$$[\mathbf{x}]_2 = \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},2})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m], [\mathbf{y}]_2 = \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},2})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n],$$

  and $[W]_2 = \mathbf{u}_2[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m] \otimes \mathbf{v}_2[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n]$.
- P_3 outputs $([\mathbf{x}]_3, [\mathbf{y}]_3, [W]_3)$, where

$$[\mathbf{x}]_3 = \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},3})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m], [\mathbf{y}]_3 = \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},3})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n],$$

  and $[W]_3 = \mathbf{u}_3[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m] \otimes \mathsf{PRG}(\mathsf{sd}_{\mathbf{y},\mathbf{d}})[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n] + \mathsf{PRG}(\mathsf{sd}_{\mathbf{x},\mathbf{b}})[\mathsf{flag}_{\mathbf{x}} + 1, ..., \mathsf{flag}_{\mathbf{x}} + m] \otimes \mathbf{v}_3[\mathsf{flag}_{\mathbf{y}} + 1, ..., \mathsf{flag}_{\mathbf{y}} + n]$.
- All three parties update the indices $(\mathsf{flag}_{\mathbf{x}}, \mathsf{flag}_{\mathbf{y}}) \leftarrow (\mathsf{flag}_{\mathbf{x}} + m, \mathsf{flag}_{\mathbf{y}} + n)$.

Fig. 5. 3PC tensor triple utilization process

In this section we provide a practical way to fulfill the functionality $\mathscr{F}_{\mathsf{3PCTT}}^{K,m,n}$ with the aid of a trusted server. The generation process is described explicitly in Fig. 4.

The distribution protocol uses a pseudo-random generator PRG. Due to the necessary property of the distribution process described in the table we may assume PRG will be further implemented using stream ciphers or block ciphers in a suitable mode of operation (CFB or CTR mode for instance) so that the PRG function is computed block-by-block and may continue to extend as the security parameter permits. The trusted server, once initializing the distribution process, will continuously generate and distribute a tensor triple of an arbitrary length until receiving a pause or stop signal. The parties can store the triple in parts as seeds and vectors without expanding them in matrix forms. In the future when the parties launch a 3PC protocol and an (m, n)-tensor triple is in need, they can trim the triple in suitable sizes and expand them according to the utilization process described in Fig. 5.

Table 1. Comparison of communication cost of 3PC tensor triple generation protocols

Communication/Storage	Total
sVOLE [5]	$8\lambda + mn \log \lvert K \rvert$
3PC Tensor Triple	$12\lambda + 2(m+n) \log \lvert K \rvert$

The main advantage of the distribution protocol is the improvement on the communication and storage cost. Compared to the subfield VOLE type distribution protocol, the communication and storage cost is considerably decreased, as shown in the Table 1. As an instance, the total storage cost for a $(1024, 1024)$-tensor triple of 32-bit elements is approximately 4MB for sVOLE but only approximately 16.2KB for 3PC tensor triples. Furthermore, the streamline construction of the distribution protocol above does not work well for sVOLE triples, since the server in this case would have to distribute a matrix which expands infinitely in two dimensions. Even if the server can manage to fulfill this job, in further use the trimming process must on average discard 50% of the entries as only the diagonal blocks can be used when trimming.

Furthermore, when a streamline process is applied, the utility rate of tensor triples is 100% since the trimming procedure is applied to an infinite (or in practice, a considerably large) dimension. This further improves the pre-computability of tensor triples. Using the server-aided protocol, the participants do not need any prior knowledge of the sizes of the matrices occurred in further protocols while being able to fully utilize all tensor triples distributed. As far as of the author's knowledge, this would be the first multi-dimensional pseudorandom correlation distribution which can achieve such a property.

4.3 4PC tensor triple generation

When four or more parties are involved. The average communication cost can be further reduced in the honest majority setting. We briefly introduce the protocol in Fig. 6.

4PC tensor triple distribution $\mathsf{TT.4PCDist}()$

Players: A trusted server S. Parties P_1, P_2, P_3, P_4.

Interacting Phase:

1. S randomly samples seeds $\mathsf{sd}_{\mathbf{x},1}, \mathsf{sd}_{\mathbf{x},2}, \mathsf{sd}_{\mathbf{x},3}, \mathsf{sd}_{\mathbf{x},4}$, $\mathsf{sd}_{\mathbf{y},1}, \mathsf{sd}_{\mathbf{y},2}, \mathsf{sd}_{\mathbf{y},3}, \mathsf{sd}_{\mathbf{y},4}$, $\mathsf{sd}_{\mathbf{x},\mathbf{a}}, \mathsf{sd}_{\mathbf{y},\mathbf{b}}$;
2. S computes $\mathbf{x} = \sum_{i=1}^4 \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},i})$, $\mathbf{y} = \sum_{i=1}^4 \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},i})$, $\mathbf{a} = \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},\mathbf{a}})$, $\mathbf{b} = \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},\mathbf{b}})$. It then computes $\mathbf{u} = \mathbf{a} + \mathbf{x}, \mathbf{v} = \mathbf{b} + \mathbf{y}$;
3. S sends $(\mathsf{sd}_{\mathbf{x},1}, \mathsf{sd}_{\mathbf{y},1}, \mathbf{u}, \mathbf{v})$ to P_1, $(\mathsf{sd}_{\mathbf{x},2}, \mathsf{sd}_{\mathbf{y},2}, \mathbf{u}, \mathsf{sd}_{\mathbf{y},\mathbf{b}})$ to P_2, $(\mathsf{sd}_{\mathbf{x},3}, \mathsf{sd}_{\mathbf{y},3}, \mathsf{sd}_{\mathbf{x},\mathbf{a}}, \mathbf{v})$ to P_3, and $(\mathsf{sd}_{\mathbf{x},4}, \mathsf{sd}_{\mathbf{y},4}, \mathsf{sd}_{\mathbf{x},\mathbf{a}}, \mathsf{sd}_{\mathbf{y},\mathbf{b}})$ to P_4.

Output:

- P_1 outputs $([\mathbf{x}]_1 = \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},1}), [\mathbf{y}]_1 = \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},1}), [W]_1 = \mathbf{u} \otimes \mathbf{v})$;
- P_2 outputs $([\mathbf{x}]_2 = \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},2}), [\mathbf{y}]_2 = \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},2}), [W]_2 = -\mathbf{u} \otimes \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},\mathbf{b}}))$;
- P_3 outputs $([\mathbf{x}]_3 = \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},3}), [\mathbf{y}]_3 = \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},3}), [W]_3 = -\mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},\mathbf{a}}) \otimes \mathbf{v})$;
- P_4 outputs $([\mathbf{x}]_4 = \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},4}), [\mathbf{y}]_4 = \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},4}), [W]_4 = \mathsf{PRG}_1(\mathsf{sd}_{\mathbf{x},\mathbf{a}}) \otimes \mathsf{PRG}_2(\mathsf{sd}_{\mathbf{y},\mathbf{b}}))$.

Fig. 6. Server-aided 4PC tensor triple distribution protocol

The correctness follows from the computation below.

$$\begin{aligned} W =&[W]_1 + [W]_2 + [W]_3 + [W]_4 \\ =&(\mathbf{a} + \mathbf{x}) \otimes (\mathbf{b} + \mathbf{y}) - (\mathbf{a} + \mathbf{x}) \otimes \mathbf{b} - \mathbf{a} \otimes (\mathbf{b} + \mathbf{y}) + \mathbf{a} \otimes \mathbf{b} \\ =&\mathbf{x} \otimes \mathbf{y} \end{aligned}$$

For security, one could obtain a similar security proposition as in the 3PC case and we omit the explicit statement for brevity.

For communicational efficiency, the protocol has a total communication cost $12\lambda + 2(m+n)\log q$ bits and a maximal communication cost $2\lambda + (m+n)\log q$ bits to a single party. This shows the 4PC protocol achieved a better communicational efficiency on average.

Similarly, in this setting we also assume W_i is never directly published. In this final annoucement procedure, either W_i is adequately amalgamated or the final MPC share is extra masked.

4.4 More than four participants

If the number of participants is greater than four, there are ways to construct a similar decomposition for each specific case, but the solutions in 3PC and 4PC cases do not have a systematic generalization. Roughly speaking, we believe the main reason is due to the lack of a secure decomposition in the 2PC setting, at least similar to ours, or more generally, the absence of a similar decomposition in the case when the adversary can collude with exactly half of the participants. Therefore, as the number of participants grows, we expect the asymptotic difficulty to construct a secure server-aided tensor triple distribution protocol actually increases as well. Hence it is not surprising the distribution protocols we propose do not attain a natural generalization.

5 Implementation

Our implementations are based on C/C++. We have used OpenSSL library for the generation process of cryptographic secure pseudorandom numbers. We use our method to generate 10,000 tensor triples of 128-bit length and of size 1024×1024 and compute the average time per triple. The experiments are run on desktop with Intel i7-14700F 2.10 GHz CPU and 16GB RAM.

For sVOLE-based 3PC tensor triple generation method we utilize the protocol introduced in [7] of a transformation from a generic 2PC simple bilinear correlation protocol to a MPC one, and then apply the implementation realized in [6] for the 2PC primitive. The comparison of the performances between the generic transformation and our method is listed in Table 2. We see our method achieves an amazingly optimization in both computation and communication costs. As a reference, we also provide a theoretical communication cost of the naive server-aided 3PC sVOLE-based tensor triple generation protocol.

Table 2. Performance of 3PC $(1024, 1024)$-tensor triple generation implementations

	sVOLE [6, 7]		sVOLE	Ours
Type	COT	SOT	Server-Aided	Server-Aided
Generation time (ms)	1914	6813	-	4.23
Communication (MB)	3084	1377	16	0.063

Also, as a comparison with [10], the authors have achieved a generation of approximately 108,000 Beaver triples of 128-bit length for three parties, while our work is capable of generating approximately 236 tensor triples for three parties of size 1024×1024 and 128-bit length. In the most extravagent case, the three parties only use the diagonal entries as Beaver triples and our protocol still provides approximately 242,000 Beaver triples of 128-bit length.

We also list the necessary offline cost in Table 3 for securely computing a matrix product of two matrices of sizes 1024×1024 over 128-bit ambient space. The cost of the generic sVOLE method is often unbearable in practice for its high memory storage cost, while our method only requires a comparatively small and acceptable cost. As a reference, we also provide theoretical communication and memory costs of the naive server-aided 3PC sVOLE-based tensor triple generation protocol.

Table 3. Offline cost of 3PC secure matrix multiplication protocol of dimensions 1024×1024 and 128-bit entry length

	sVOLE [6, 7]		sVOLE	Ours
Type	COT	SOT	Server-Aided	Server-Aided
Generation time (s)	1960	6977	-	4.4
Communication (GB)	3084	1377	16	0.063
Memory storage (GB)	24600	24600	48	48

As a remark, we see the importing of a server results in a significant raise in efficiency, both communicatively and computationally.

6 Conclusion

Tensor triple is a new kind of correlation which is very suitable for multi-dimensional MPC. It can be used to accelerate many existing privacy-preserving biometric idenfication protocols and privacy-preserving machine learning protocols which mainly involve vector and matrix operations. Our method provides an extremely efficient way to generate and distribute tensor triples in 3PC and 4PC honest majority settings. This further enables the capability of tensor triple to provide masks for high dimensional matrix objects and facilitate the MPC protocols in which large matrices together with their interactions are heavily involved, such as privacy-preserving machine learning protocols. In follow-up works we will also apply the server-aided tensor triple distribution protocols to assist various privacy-preserving machine learning protocols.

References

1. M. Abspoel, R. Cramer, I. Damgrard, D. Escudero, M. Rambaud, C. Xing, and C. Yuan, "Asymptotically good multiplicative LSSS over galois rings and applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$," in *Advances in Cryptology - ASIACRYPT 2020*, pp. 151–180, 2020.
2. M. Naor and B. Pinkas, "Oblivious polynomial evaluation," *SIAM Journal on Computing*, vol. 35, no. 5, pp. 1254–1281, 2006.
3. B. Applebaum, I. Damgrard, Y. Ishai, M. Nielsen, and L. Zichron, "Secure arithmetic computation with constant computational overhead," in *Advances in Cryptology – CRYPTO 2017* (J. Katz and H. Shacham, eds.), pp. 223–254, 2017.
4. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector ole," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, p. 896–912, 2018.
5. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, p. 291–308, 2019.
6. D. Wu, B. Liang, Z. Lu, and J. Ding, "Efficient secure multi-party computation for multi-dimensional arithmetics and its application in privacy-preserving biometric identification," in *Cryptology and Network Security (CANS)*, 2024.
7. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent ot extension and more," in *Advances in Cryptology – CRYPTO 2019* (A. Boldyreva and D. Micciancio, eds.), (Cham), pp. 489–518, Springer International Publishing, 2019.
8. D. Beaver, "Commodity-based cryptography (extended abstract)," in *Symposium on the Theory of Computing*, 1997.
9. N. P. Smart and T. Tanguy, "Taas: Commodity mpc via triples-as-a-service," *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019.
10. P. Muth and S. Katzenbeisser, "Assisted mpc." Cryptology ePrint Archive, Paper 2022/1453, 2022.
11. Z. Qiu, K. Yang, Y. Yu, and L. Zhou, "Maliciously secure multi-party psi with lower bandwidth and faster computation," in *Information and Communications Security* (C. Alcaraz, L. Chen, S. Li, and P. Samarati, eds.), (Cham), pp. 69–88, Springer International Publishing, 2022.
12. S. Zhang, "Efficient VOLE based multi-party PSI with lower communication cost." Cryptology ePrint Archive, Paper 2023/1690, 2023.
13. M. O. Rabin, "How to exchange secrets with oblivious transfer," *TR-81 edition*, 1981.
14. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology - CRYPTO 2003*, pp. 145–161, 2003.
15. V. Kolesnikov and R. Kumaresan, "Improved ot extension for transferring short secrets," in *Advances in Cryptology – CRYPTO 2013* (R. Canetti and J. A. Garay, eds.), pp. 54–70, 2013.

16. V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, p. 818–829, 2016.
17. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, p. 535–548, 2013.
18. K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated ot with small communication," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, p. 1607–1626, 2020.
19. P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, "Distributed vector-ole: Improved constructions and implementation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, p. 1055–1072, 2019.
20. G. Couteau, P. Rindal, and S. Raghuraman, "Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes," in *Advances in Cryptology – CRYPTO 2021*, pp. 502–534, 2021.
21. S. Raghuraman, P. Rindal, and T. Tanguy, "Expand-convolute codes for pseudorandom correlation generators from lpn," in *Advances in Cryptology – CRYPTO 2023*, pp. 602–632, 2023.
22. R. Cramer, I. Damgrard, and Y. Ishai, "Share conversion, pseudorandom secret-sharing and applications to secure computation," in *Theory of Cryptography* (J. Kilian, ed.), (Berlin, Heidelberg), pp. 342–362, Springer Berlin Heidelberg, 2005.

A Proof of masking

In chapter 3 we introduced the two types of security of our tensor triple distribution protocol in Theorem 3 and Theorem 4. We provide proofs of the statements in this section.

Proof of theorem 3. We prove by cases.

1. P_1 is corrupted:
   The simulator receives $(\mathbf{x}_1, \mathbf{y}_1, \mathbf{x}'_1, \mathbf{y}'_1)$ from $\mathcal{A}$. It flips a fair coin to determine his choice of $(\mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}})$. It then samples
   $$\mathbf{x}_2^{\text{sim}}, \mathbf{x}_3^{\text{sim}} \leftarrow_\$ K^m, \quad \mathbf{y}_2^{\text{sim}}, \mathbf{y}_3^{\text{sim}} \leftarrow_\$ K^n.$$
   Then it computes
   $$\mathbf{x}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{x}_i^{\text{sim}}, \quad \mathbf{y}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{y}_i^{\text{sim}}.$$
   It then samples
   $$\mathbf{a}_1^{\text{sim}}, \mathbf{b}_1^{\text{sim}} \leftarrow_\$ K^m, \quad \mathbf{c}_1^{\text{sim}}, \mathbf{d}_1^{\text{sim}} \leftarrow_\$ K^n.$$
   It then computes and sets
   $$\mathbf{b}_2^{\text{sim}} = \mathbf{b}_1^{\text{sim}} + \mathbf{x}^{\text{sim}}, \quad \mathbf{d}_2^{\text{sim}} = \mathbf{d}_1^{\text{sim}} + \mathbf{y}^{\text{sim}},$$
   and
   $$\mathbf{a}_3^{\text{sim}} = \mathbf{a}_1^{\text{sim}} - \mathbf{x}^{\text{sim}}, \quad \mathbf{c}_3^{\text{sim}} = \mathbf{c}_1^{\text{sim}} - \mathbf{y}^{\text{sim}}$$
   as well as $\mathbf{b}_3^{\text{sim}} = \mathbf{b}_1^{\text{sim}}$ and $\mathbf{d}_3^{\text{sim}} = \mathbf{d}_1^{\text{sim}}$. At last the simulator outputs the following:
   - $(\mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}}, \mathbf{a}_1^{\text{sim}}, \mathbf{b}_1^{\text{sim}}, \mathbf{c}_1^{\text{sim}}, \mathbf{d}_1^{\text{sim}})$ to P_1,
   - $(\mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}}, \mathbf{b}_2^{\text{sim}}, \mathbf{d}_2^{\text{sim}})$ to P_2 and

   – $(\mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}}, \mathbf{a}_3^{\text{sim}}, \mathbf{b}_3^{\text{sim}}, \mathbf{c}_3^{\text{sim}}, \mathbf{d}_3^{\text{sim}})$ to P_3.

   The simulator above simulates perfectly the functionality and it can seen the joint probability distribution of
$$(\mathbf{a}_1^{\text{sim}}, \mathbf{b}_1^{\text{sim}}, \mathbf{c}_1^{\text{sim}}, \mathbf{d}_1^{\text{sim}}, \mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}}, \mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}})$$
   is independent of the distribution of $(\mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}})$.

2. P_2 is corrupted:
   The simulator receives $(\mathbf{x}_2, \mathbf{y}_2, \mathbf{x}_2', \mathbf{y}_2')$ from $\mathcal{A}$. It flips a fair coin to determine his choice of $(\mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}})$. It then samples
$$\mathbf{x}_1^{\text{sim}}, \mathbf{x}_3^{\text{sim}} \leftarrow_{\$} K^m, \quad \mathbf{y}_1^{\text{sim}}, \mathbf{y}_3^{\text{sim}} \leftarrow_{\$} K^n.$$
   Then it computes
$$\mathbf{x}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{x}_i^{\text{sim}}, \quad \mathbf{y}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{y}_i^{\text{sim}}.$$
   It then samples
$$\mathbf{a}_1^{\text{sim}}, \mathbf{b}_2^{\text{sim}} \leftarrow_{\$} K^m, \quad \mathbf{c}_1^{\text{sim}}, \mathbf{d}_2^{\text{sim}} \leftarrow_{\$} K^n.$$
   It then computes and sets
$$\mathbf{b}_1^{\text{sim}} = \mathbf{b}_2^{\text{sim}} - \mathbf{x}^{\text{sim}}, \quad \mathbf{d}_1^{\text{sim}} = \mathbf{d}_2^{\text{sim}} - \mathbf{y}^{\text{sim}},$$
   and
$$\mathbf{a}_3^{\text{sim}} = \mathbf{a}_1^{\text{sim}} - \mathbf{x}^{\text{sim}}, \quad \mathbf{c}_3^{\text{sim}} = \mathbf{c}_1^{\text{sim}} - \mathbf{y}^{\text{sim}}$$
   as well as $\mathbf{b}_3^{\text{sim}} = \mathbf{b}_1^{\text{sim}}$ and $\mathbf{d}_3^{\text{sim}} = \mathbf{d}_1^{\text{sim}}$. At last the simulator outputs the following:
   – $(\mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}}, \mathbf{a}_1^{\text{sim}}, \mathbf{b}_1^{\text{sim}}, \mathbf{c}_1^{\text{sim}}, \mathbf{d}_1^{\text{sim}})$ to P_1,
   – $(\mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}}, \mathbf{b}_2^{\text{sim}}, \mathbf{d}_2^{\text{sim}})$ to P_2 and
   – $(\mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}}, \mathbf{a}_3^{\text{sim}}, \mathbf{b}_3^{\text{sim}}, \mathbf{c}_3^{\text{sim}}, \mathbf{d}_3^{\text{sim}})$ to P_3.

   The simulator above simulates perfectly the functionality and it can seen the joint probability distribution of
$$(\mathbf{b}_2^{\text{sim}}, \mathbf{d}_2^{\text{sim}}, \mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}}, \mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}})$$
   is independent of the distribution of $(\mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}})$.

3. P_3 is corrupted:
   The simulator receives $(\mathbf{x}_3, \mathbf{y}_3, \mathbf{x}_3', \mathbf{y}_3')$ from $\mathcal{A}$. It flips a fair coin to determine his choice of $(\mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}})$. It then samples
$$\mathbf{x}_1^{\text{sim}}, \mathbf{x}_2^{\text{sim}} \leftarrow_{\$} K^m, \quad \mathbf{y}_1^{\text{sim}}, \mathbf{y}_2^{\text{sim}} \leftarrow_{\$} K^n.$$
   Then it computes
$$\mathbf{x}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{x}_i^{\text{sim}}, \quad \mathbf{y}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{y}_i^{\text{sim}}.$$
   It then samples
$$\mathbf{a}_3^{\text{sim}}, \mathbf{b}_3^{\text{sim}} \leftarrow_{\$} K^m, \quad \mathbf{c}_3^{\text{sim}}, \mathbf{d}_3^{\text{sim}} \leftarrow_{\$} K^n.$$
   It then computes and sets
$$\mathbf{b}_2^{\text{sim}} = \mathbf{b}_3^{\text{sim}} + \mathbf{x}^{\text{sim}}, \quad \mathbf{d}_2^{\text{sim}} = \mathbf{d}_3^{\text{sim}} + \mathbf{y}^{\text{sim}},$$
   and
$$\mathbf{a}_1^{\text{sim}} = \mathbf{a}_3^{\text{sim}} + \mathbf{x}^{\text{sim}}, \quad \mathbf{c}_1^{\text{sim}} = \mathbf{c}_3^{\text{sim}} + \mathbf{y}^{\text{sim}}$$
   as well as $\mathbf{b}_1^{\text{sim}} = \mathbf{b}_3^{\text{sim}}$ and $\mathbf{d}_1^{\text{sim}} = \mathbf{d}_3^{\text{sim}}$. At last the simulator outputs the following:

- $(\mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}}, \mathbf{a}_1^{\text{sim}}, \mathbf{b}_1^{\text{sim}}, \mathbf{c}_1^{\text{sim}}, \mathbf{d}_1^{\text{sim}})$ to P_1,
- $(\mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}}, \mathbf{b}_2^{\text{sim}}, \mathbf{d}_2^{\text{sim}})$ to P_2 and
- $(\mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}}, \mathbf{a}_3^{\text{sim}}, \mathbf{b}_3^{\text{sim}}, \mathbf{c}_3^{\text{sim}}, \mathbf{d}_3^{\text{sim}})$ to P_3.

The simulator above simulates perfectly the functionality and it can seen the joint probability distribution of

$$(\mathbf{a}_3^{\text{sim}}, \mathbf{b}_3^{\text{sim}}, \mathbf{c}_3^{\text{sim}}, \mathbf{d}_3^{\text{sim}}, \mathbf{x}_1^{\text{sim}}, \mathbf{y}_1^{\text{sim}}, \mathbf{x}_2^{\text{sim}}, \mathbf{y}_2^{\text{sim}})$$

is independent of the distribution of $(\mathbf{x}_3^{\text{sim}}, \mathbf{y}_3^{\text{sim}})$.

□

Proof of theorem 4. We prove by showing the security can be reduced to the security of a generic simple bilinear correlation protocol introduced in [7]. We only prove the case where P_1 is corrupted and the proof for the rest of the cases is similar.

For the l-th call of the functionality $\mathscr{F}_{\mathsf{3PCTT}}^{K,n,n}$, the simulator receives $(\mathbf{x}_{1,l}, \mathbf{y}_{1,l}, \mathbf{x}'_{1,l}, \mathbf{y}'_{1,l})$ from $\mathcal{A}$. It flips a fair coin to determine his choice of $(\mathbf{x}_{1,l}^{\text{sim}}, \mathbf{y}_{1,l}^{\text{sim}})$. It then samples

$$\mathbf{x}_{2,l}^{\text{sim}}, \mathbf{x}_{3,l}^{\text{sim}} \leftarrow_{\$} K^m, \quad \mathbf{y}_{2,l}^{\text{sim}}, \mathbf{y}_{3,l}^{\text{sim}} \leftarrow_{\$} K^n.$$

Then it computes

$$\mathbf{x}_{\text{full},l}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{x}_{i,l}^{\text{sim}}, \quad \mathbf{y}_{\text{full},l}^{\text{sim}} = \sum_{i=1}^{3} \mathbf{y}_{i,l}^{\text{sim}}.$$

It then samples

$$\mathbf{a}_{1,l}^{\text{sim}}, \mathbf{b}_{1,l}^{\text{sim}} \leftarrow_{\$} K^m, \quad \mathbf{c}_{1,l}^{\text{sim}}, \mathbf{d}_{1,l}^{\text{sim}} \leftarrow_{\$} K^n.$$

It then computes and sets

$$\mathbf{b}_{2,l}^{\text{sim}} = \mathbf{b}_{1,l}^{\text{sim}} + \mathbf{x}_{\text{full},l}^{\text{sim}}, \quad \mathbf{d}_{2,l}^{\text{sim}} = \mathbf{d}_{1,l}^{\text{sim}} + \mathbf{y}_{\text{full},l}^{\text{sim}},$$

and

$$\mathbf{a}_{3,l}^{\text{sim}} = \mathbf{a}_{1,l}^{\text{sim}} - \mathbf{x}_{\text{full},l}^{\text{sim}}, \quad \mathbf{c}_{3,l}^{\text{sim}} = \mathbf{c}_{1,l}^{\text{sim}} - \mathbf{y}_{\text{full},l}^{\text{sim}}$$

as well as $\mathbf{b}_{3,l}^{\text{sim}} = \mathbf{b}_{1,l}^{\text{sim}}$ and $\mathbf{d}_{3,l}^{\text{sim}} = \mathbf{d}_{1,l}^{\text{sim}}$. At last the simulator outputs the following:

- $(\mathbf{x}_{1,l}^{\text{sim}}, \mathbf{y}_{1,l}^{\text{sim}}, \mathbf{a}_{1,l}^{\text{sim}}, \mathbf{b}_{1,l}^{\text{sim}}, \mathbf{c}_{1,l}^{\text{sim}}, \mathbf{d}_{1,l}^{\text{sim}})$ to P_1,
- $(\mathbf{x}_{2,l}^{\text{sim}}, \mathbf{y}_{2,l}^{\text{sim}}, \mathbf{b}_{2,l}^{\text{sim}}, \mathbf{d}_{2,l}^{\text{sim}})$ to P_2 and
- $(\mathbf{x}_{3,l}^{\text{sim}}, \mathbf{y}_{3,l}^{\text{sim}}, \mathbf{a}_{3,l}^{\text{sim}}, \mathbf{b}_{3,l}^{\text{sim}}, \mathbf{c}_{3,l}^{\text{sim}}, \mathbf{d}_{3,l}^{\text{sim}})$ to P_3.

The simulator above simulates perfectly the functionality. Now note that

$$W_2 = (B + X)(D + Y),$$

$$W_3 = (A - X)D + B(C - Y),$$

where

$$A = (\mathbf{a}_{1,l}^T)_{l=1}^n, B = (\mathbf{b}_{1,l}^T)_{l=1}^n,$$
$$C = (\mathbf{c}_{1,l})_{l=1}^n, D = (\mathbf{d}_{1,l})_{l=1}^n,$$

and

$$X = (\mathbf{x}_{\text{full},l}^T)_{l=1}^n, Y = (\mathbf{y}_{\text{full},l})_{l=1}^n.$$

The adversary $\mathcal{A}$ will obtain

$$\begin{array}{l}(\{\mathbf{a}_{1,l}, \mathbf{b}_{1,l}, \mathbf{c}_{1,l}, \mathbf{d}_{1,l},\\ \mathbf{x}_{2,l}, \mathbf{y}_{2,l}, \mathbf{x}_{3,l}, \mathbf{y}_{3,l}\}, W_2, W_3)\end{array}$$

Assume $\mathcal{A}$ has a probabilistic polynomial time algorithm which is able to distinguish the two settings

$$\left[\begin{array}{c}(\{\mathbf{x}_{1,l}, \mathbf{y}_{1,l}, \mathbf{x}'_{1,l}, \mathbf{y}'_{1,l}\}) \leftarrow \mathcal{A}(1^\lambda),\\ (\{\mathbf{a}_{1,l}, \mathbf{b}_{1,l}, \mathbf{c}_{1,l}, \mathbf{d}_{1,l}, \mathbf{x}_{2,l}, \mathbf{x}_{3,l},\\ \mathbf{b}_{2,l}, \mathbf{a}_{3,l}, \mathbf{b}_{3,l}, \mathbf{y}_{2,l}, \mathbf{y}_{3,l}, \mathbf{d}_{2,l}, \mathbf{c}_{3,l}, \mathbf{d}_{3,l}\})\\ \leftarrow \mathscr{F}^{K,n,n}_{\mathsf{3PCTT}}(\{\mathbf{x}_{1,l}, \mathbf{y}_{1,l}\}, \mathsf{Corrupt} = 1)\\ : \mathcal{A} \leftarrow (\{\mathbf{a}_{1,l}, \mathbf{b}_{1,l}, \mathbf{c}_{1,l}, \mathbf{d}_{1,l},\\ \mathbf{x}_{2,l}, \mathbf{y}_{2,l}, \mathbf{x}_{3,l}, \mathbf{y}_{3,l}\}, W_2, W_3)\\ (l = 1, ..., n)\end{array}\right]$$

and

$$\left[\begin{array}{c}(\{\mathbf{x}_{1,l}, \mathbf{y}_{1,l}, \mathbf{x}'_{1,l}, \mathbf{y}'_{1,l}\}) \leftarrow \mathcal{A}(1^\lambda),\\ (\{\mathbf{a}_{1,l}, \mathbf{b}_{1,l}, \mathbf{c}_{1,l}, \mathbf{d}_{1,l}, \mathbf{x}_{2,l}, \mathbf{x}_{3,l},\\ \mathbf{b}_{2,l}, \mathbf{a}_{3,l}, \mathbf{b}_{3,l}, \mathbf{y}_{2,l}, \mathbf{y}_{3,l}, \mathbf{d}_{2,l}, \mathbf{c}_{3,l}, \mathbf{d}_{3,l}\})\\ \leftarrow \mathscr{F}^{K,n,n}_{\mathsf{3PCTT}}(\{\mathbf{x}'_{1,l}, \mathbf{y}'_{1,l}\}, \mathsf{Corrupt} = 1)\\ : \mathcal{A} \leftarrow (\{\mathbf{a}_{1,l}, \mathbf{b}_{1,l}, \mathbf{c}_{1,l}, \mathbf{d}_{1,l},\\ \mathbf{x}_{2,l}, \mathbf{y}_{2,l}, \mathbf{x}_{3,l}, \mathbf{y}_{3,l}\}, W_2, W_3)\\ (l = 1, ..., n)\end{array}\right]$$

which we denote by A_1 with a non-negligible advantage. Then we consider a simple bilinear correlation functionality $\mathscr{F}^{K,n,n}_{\mathsf{SBC}}$ to provide

$$\begin{aligned}(\mathbf{u}_i, x_i, \mathbf{v}_i) \leftarrow_{\$} & \mathbb{K}^n \times \mathbb{F} \times \mathbb{F}^n\\ \cong & \mathbb{K}^n \times \mathbb{K}^n \times \mathbb{K}^{n\times n}.\end{aligned}$$

$\mathcal{A}$ now calls $\mathscr{F}^{K,n,n}_{\mathsf{SBC}}$ for n times and colludes with P_1. Consider the canonical isomorphism of vectors $\varphi : \mathbb{F} \to \mathbb{K}^n$. For instance,

$$\varphi(a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1}) = (a_0, ..., a_{n-1})$$

where $\mathbb{F}$ is realized through an extension $\mathbb{K}[\alpha]$ by adding a root of an irreducible monic polynomial of degree n. $\mathcal{A}$ sends to the simulator

$$(\mathbf{u}_{1,l}, x_{1,l}) = (\mathbf{x}_{1,l}, \varphi^{-1}(\mathbf{y}_{1,l}))$$

and

$$(\mathbf{u}'_{1,l}, x'_{1,l}) = (\mathbf{x}'_{1,l}, \varphi^{-1}(\mathbf{y}'_{1,l}))$$

for $1 \le l \le n$.

Suppose for the l-th call of $\mathscr{F}^{K,n,n}_{\mathsf{SBC}}$, the simulator returns $(\mathbf{u}_{i,l}, x_{i,l}, \mathbf{v}_{i,l})$ to P_i. According to the security assumption in [7]. $\mathcal{A}$ has access to

$$(\mathbf{u}_{2,l}, x_{2,l}, \mathbf{v}_{2,l}, \mathbf{u}_{3,l}, x_{3,l}, \mathbf{v}_{3,l}).$$

Now $\mathcal{A}$ input to A_1 the following data:

$$\begin{aligned}&(\mathbf{x}_{2,l} = \mathbf{u}_{2,l}, \mathbf{y}_{2,l} = \varphi^{-1}(x_{2,l}),\\ &\mathbf{x}_{3,l} = \mathbf{u}_{3,l}, \mathbf{y}_{3,l} = \varphi^{-1}(x_{3,l}),\\ &W_2 = \varphi^{-1}(\sum_{l=1}^{n} \mathbf{v}_{2,l}), W_3 = \varphi^{-1}(\sum_{l=1}^{n} \mathbf{v}_{3,l}),\\ &W_{1,l} = \varphi^{-1}(\mathbf{v}_{1,l})).\end{aligned}$$

Note that W_2 and W_3 are computationally indistinguishable from two uniformly random matrices (B, D randomizes W_2 and then A, C randomizes W_3). Hence the distributions of W_i and $\varphi^{-1}(\sum_{l=1}^{n} \mathbf{v}_{i,l})$ can be considered identical computationally. Then $\mathcal{A}$ obtains a non-negligible advantage on distinguishing the two settings

$$\begin{bmatrix} (\{\mathbf{u}_{1,l}, x_{1,l}, \mathbf{u}'_{1,l}, x'_{1,l}\}) \leftarrow \mathcal{A}(1^\lambda), \\ (\{\mathbf{u}_{i,l}, x_{i,l}, \mathbf{v}_{i,l}\}) \\ \leftarrow \mathscr{F}_{\mathsf{SBC}}^{K,n,n}(\{\mathbf{u}_{1,l}, x_{1,l}\}, \mathsf{Corrupt} = 1) \\ : \mathcal{A} \leftarrow (\{\mathbf{u}_{2,l}, x_{2,l}, \mathbf{v}_{2,l}, \\ \mathbf{u}_{3,l}, x_{3,l}, \mathbf{v}_{3,l}, \mathbf{v}_{1,l}\}) \\ (l = 1, ..., n) \end{bmatrix}$$

and

$$\begin{bmatrix} (\{\mathbf{u}_{1,l}, x_{1,l}, \mathbf{u}'_{1,l}, x'_{1,l}\}) \leftarrow \mathcal{A}(1^\lambda), \\ (\{\mathbf{u}_{i,l}, x_{i,l}, \mathbf{v}_{i,l}\}) \\ \leftarrow \mathscr{F}_{\mathsf{SBC}}^{K,n,n}(\{\mathbf{u}'_{1,l}, x'_{1,l}\}, \mathsf{Corrupt} = 1) \\ : \mathcal{A} \leftarrow (\{\mathbf{u}_{2,l}, x_{2,l}, \mathbf{v}_{2,l}, \\ \mathbf{u}_{3,l}, x_{3,l}, \mathbf{v}_{3,l}, \mathbf{v}_{1,l}\}) \\ (l = 1, ..., n) \end{bmatrix}$$

This gives a probabilistic polynomial time algorithm to break the functionality $\mathscr{F}_{\mathsf{SBC}}^{K,n,n}$ given an oracle distinguishing settings in $\mathscr{F}_{\mathsf{3PCTT}}^{K,n,n}$. This finishes the security reduction for a colluded P_1. $\square$