# Efficient Secure Multiparty Computation for Multidimensional Arithmetics and Its Application in Privacy-Preserving Biometric Identification

Dongyu Wu, Bei Liang, Zijie Lu and Jintai Ding

Beijing Institute of Mathematical Sciences and Applications

**Abstract.** Over years of the development of secure multi-party computation (MPC), many sophisticated functionalities have been made pratical and multi-dimensional operations occur more and more frequently in MPC protocols, especially in protocols involving datasets of vector elements, such as privacy-preserving biometric identification and privacy-preserving machine learning. In this paper, we introduce a new kind of correlation, called tensor triples, which is designed to make multi-dimensional MPC protocols more efficient. We will discuss the generation process, the usage, as well as the applications of tensor triples and show that it can accelerate privacy-preserving biometric identification protocols, such as FingerCode, Eigenfaces and FaceNet, by more than 1000 times.

**Keywords:** Tensor triple · MPC · Beaver triple · VOLE · Privacy-preserving biometric identification · Privacy-preserving machine learning

## 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. Starting from Yao's Garbled Circuit [Yao86], numerous advances have been made on the most fundamental circuit-based MPC. Over years of development, circuit-based MPC gradually gains capability of practically computing more and more sophisticated and large-scale functions. Apart from one-dimensional numeric computation, the necessity of handling higher dimensional operations has also become a concern. For instance, people for years have explored the possibility of application of MPC on privacy-preserving biometric identification [EFG+09, OPJM10, SSW10, BG11, HMEK11, BFCP12, LCP+12, LPS12, SSNO12, WK12, BCP13a, BCP13b, BCF+14, BEOMC16, DCH+16, GBFG+16, HH16, GBGMF17, MLL+19], privacy-preserving machine learning [NWI+13, BIK+17, MR18, MRT20, KPPS21] and many other practical fields that require an intensive use of vector and matrix operations, such as vector tensoring and matrix multiplication. Unlike the straight-forward homomorphic encryption methods, which often behave less computationally efficient, it is crucial to find a way to accelerate multi-dimensional arithmetics for circuit-based MPC protocols. In recent years, apart from the classical correlation generation protocols such as OT and Beaver triples, researchers have also attempted to find new kinds of correlations that can fulfill different needs of MPC protocols and perform more efficiently compared to classical ways [NP06, ADI+17, BCGI18, BCG+19]. Among these variants, VOLE, 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. Therefore, we would like to explore a way to boost MPC protocols involving multi-dimensional objects, such as vectors and matrices, and we would like to take advantage of the correlation properties of VOLE to assist the procedure.

## 1.2 Main contribution

The goal of this paper is to explore a more efficient way to fulfill multi-dimensional secure multi-party computation. In this paper, we define a new kind of correlated triples, called tensor triples, which can be generated using VOLE, or alternatively RLWE-based homomorphic encryption. We will explain in detail how tensor triples can be generated efficiently in several ways, making them "cheap" to generate and "convenient" to use. We will also prove that this type of triples can be used to assist the secure multi-party computation on multi-dimensional arithmetics and accelerate MPC protocols involving vectors and matrices. The corresponding protocols will be much more efficient both computationally and communicatively compared to the usual Beaver's method. As a result, we discover several practical applications of tensor triples on classical privacy-preserving biometric identification and privacy-preserving machine learning protocols. As instances, we realize implementations of three privacy-preserving biometric identification protocols, namely FingerCode [JPHP99], Eigenfaces [TP91] and FaceNet [SKP15]. We will also analyze the performance of the implementations and prove our claim that the tensor triple method indeed provides a more efficient way to carry out multi-dimensional MPC protocols.

## 1.3 Arrangement of paper

In Chapter 2, we introduce define necessary notions and primitives in MPC. In Chapter 3, we define the notion of tensor triple and briefly explain the intuition and the idea how it can be used to accelerate multi-dimensional circuit-based MPC protocols. We then proceeding introducing several ways to generate tensor triples in Chapter 4. The protocols are focused on resolving the generation of tensor triples for two parties, as is the usual need in most practical applications. In Chapter 5 we explain the usage of tensor triples to accomplish MPC of basic algebraic operations for scalars, vectors and matrices. Matrix operations, especially matrix multiplication, are fundamental operations in biometric identification and machine learning, and the use of tensor triples can make these operations considerably more efficient. We will also explain in detail in the chapter the advantage of tensor triples compared to other MPC protocols fulfilling multi-dimensional operations. In Chapter 6, we discuss possible applications of tensor triples in pratice. It should be emphasized that tensor triples can be used to accelerate all multi-dimensional MPC protocols universally, and the applications we list in the chapter are merely few instances. Tensor triples can be used to efficiently perform all kinds of MPC protocols in fields such as privacy-preserving biometric identification and privacy-preserving machine learning. In Chapter 7 implementations of protocols as well as results of the experiments after the execution are presented. It can been clearly seen that the tensor triple method provide a significant speedup.

## 1.4 Security model

All protocols involved in the rest of the paper are secure against semi-honest and computationally bounded adversaries. Such an adversary will follow the protocol faithfully, but may try to learn information from what it sees during the protocol procedure.

## Acknowledgement

This work is supported by National Key R&D Program of China (No. 2021YFB2701304).

## 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 [Rab81], 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 $\mathrm{OT}_l^n$, 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 [IKNP03, KK13, KKRT16]) which generates $n$ message pairs of bit-length $l$ is denoted by $\mathrm{ROT}_l^n$. A correlated OT (COT) [ALSZ13, BCG+19, YWL+20] is a variant of ROT that allows the sender to pre-determine a string $\Delta$ and obtain two correlated random strings as the message pair with their XOR equal $\Delta$. The extension of COT denoted by $\mathrm{COT}_m^n$ 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. A subfield vector oblivious linear evaluation (sVOLE) protocol is a generalization of $\mathrm{COT}_m^n$ 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 $\mathrm{COT}_m^n$ and sVOLE functionalities defined below. It is not difficult to see that $\mathrm{RCOT}_m^n$ is a special case of $\mathscr{F}_{\mathrm{RsVOLE}}^{p,m,n}$ when $p = 2$.

Functionality $\mathrm{RCOT}_m^n$

**Players:** The sender $S$ and the receiver $R$.

**Inputs:** None.

**Outputs:**

- $S$ outputs $\mathbf{v} \in \mathbb{F}_{2^m}^n, \Delta \in \mathbb{F}_{2^m}$;
- $R$ outputs $\mathbf{u} \in \mathbb{F}_2^n, \mathbf{v} \oplus (\mathbf{u} \cdot \Delta) \in \mathbb{F}_{2^m}^n$.

Functionality $\mathscr{F}_{\mathrm{RsVOLE}}^{p,m,n}$

**Players:** The sender $S$ and the receiver $R$.

**Inputs:** A dimension pair $(n, m)$.

**Outputs:**

- $S$ outputs $x \in \mathbb{F}_{p^m}, \mathbf{v} \in \mathbb{F}_{p^m}^n$;
- $R$ outputs $\mathbf{u} \in \mathbb{F}_p^n, x\mathbf{u} + \mathbf{v} \in \mathbb{F}_{p^m}^n$.

VOLE protocols with semi-honest and computational security in OT-hybrid model have been defined in [ADI+17, BCG+19]. Subfield VOLE, as an important variant of VOLE, has also been studied and implemented in various ways (See [BCGI18, BCG+19, SGRR19, YWL+20, CRR21, RRT23]).

### 2.2 RLWE-based additively homomorphic encryption (RLWE-AHE)

We first define the notion of an AHE scheme. The definition is basically the same as the one in [RSS19], with an addition of a tensorial scalar operation which is needed for further use.

**Definition 2.1.** An AHE scheme is a tuple of algorithms AHE=(Gen, Enc, Dec, Add, ScMult, ScTensor) described as follows:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{pk},\mathsf{sk})$: Key Generation is a randomized algorithm that outputs a pair of keys $(\mathsf{pk},\mathsf{sk})$, with public key $\mathsf{pk}$ and secret key $\mathsf{sk}$.
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$: Encryption is a randomized algorithm that takes a message $m \in \mathsf{PT}_{n,\lambda}$ and the public key $\mathsf{pk}$ as input, and outputs a ciphertext $\mathsf{ct} \in \mathsf{CT}_{n,\lambda}$, where $\mathsf{PT}_{n,\lambda}$ denotes the plaintext space of $\mathsf{AHE}$ for security parameter $\lambda$ and rank $n$, and $\mathsf{CT}_{n,\lambda}$ the corresponding ciphertext space.
- $\mathsf{Dec}(\mathsf{sk},\mathsf{ct}) \to m$: Decryption is a deterministic algorithm that takes the secret key $\mathsf{sk}$ and a ciphertext $\mathsf{ct}$ and outputs the plaintext $m$.
- $\mathsf{Add}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}_+$: Addition takes two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ and the public key $\mathsf{pk}$ as input, and outputs a ciphertext $\mathsf{ct}_+ \in \mathsf{CT}_{n,\lambda}$. This binary operation with respect to $\mathsf{ct}_1, \mathsf{ct}_2$ will be denoted by $+$.
- $\mathsf{ScMult}(\mathsf{pk},\mathsf{ct}, s) \to \mathsf{ct}_\bullet$: Scalar Multiplication takes a ciphertext $\mathsf{ct} \in \mathsf{CT}_{n,\lambda}$, a plaintext $s \in \mathsf{PT}_{n,\lambda}$ and the public key $\mathsf{pk}$ as input, and outputs a ciphertext $\mathsf{ct}_\bullet \in \mathsf{CT}_{n,\lambda}$. This binary operation with respect to $\mathsf{ct}$ and $s$ will be denoted by $\bullet$.
- $\mathsf{ScTensor}(\mathsf{pk},\mathsf{ct}, \vec{s}) \to \vec{\mathsf{ct}}_\otimes$: Scalar Tensor takes a ciphertext $\mathsf{ct} \in \mathsf{CT}_{n,\lambda}$, a constant vector $\vec{s}$ of dimension $l$ and the public key $\mathsf{pk}$ as input, and outputs an array of ciphertexts $\vec{\mathsf{ct}}_\otimes \in \mathsf{CT}^l_{n,\lambda}$. This binary operation with respect to $\mathsf{ct}$ and $s$ will be denoted by $s \otimes \mathsf{ct}$.

The algorithms should satisfy the following properties:

1. Correctness:
   - For a generic pair of keys $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and any message $m$, with an overwhelming probability we have
   $$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$$
   - For a generic pair of keys $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and any two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, with an overwhelming probability we have
   $$\mathsf{Dec}(\mathsf{sk}, \mathsf{Add}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) + \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_2)$$
   - For a generic pair of keys $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, a ciphertext $\mathsf{ct}$ and a plaintext scalar $s$, with an overwhelming probability we have
   $$\mathsf{Dec}(\mathsf{sk}, \mathsf{ScMult}(\mathsf{pk}, \mathsf{ct}, s)) = s\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$$
   - For a generic pair of keys $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, a ciphertext $\mathsf{ct}$ and a scalar vector $\vec{s}$, with an overwhelming probability we have
   $$\mathsf{Dec}(\mathsf{sk}, \mathsf{ScTensor}(\mathsf{pk}, \mathsf{ct}, \vec{s})) = \vec{s} \otimes \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}),$$
   where the decryption procedure is applied to each row of the ciphertext array.
2. Security:
   The scheme is required to be IND-CPA secure.

Instances of an AHE scheme which are not based on RLWE are [Pai99, DJ01, DGK07]. While most RLWE-based instantiations of such a scheme (such as [BGV12, FV12]) are designed to be fully homomorphic, we emphasize that the additive homomorphicity suffices the scheme. We assume that the parameters of a RLWE-based AHE scheme have been chosen to be large enough to allow evaluation of the circuit for further protocols and prevent leakage through amplified ciphertext noise from homomorphic operations.

## 3 Tensor triple

In order to perform multi-dimensional multi-party computation, we first introduce the notion of vector sharing.

### 3.1 Vector sharing

**Definition 3.1.** An $(s,t)$-secret sharing of a vector $\mathbf{v}$ is a set $\{\mathbf{v}_1, ..., \mathbf{v}_s\}$ of $s$ vectors, such that there exists an efficient algorithm to reconstruct $\mathbf{v}$ from any $t$ shares $\mathbf{v}_{i_1}, ..., \mathbf{v}_{i_t}$, but there is no algorithm that can efficiently reconstruct $\mathbf{v}$ from any subset of shares with fewer cardinality. We will denote a share of $\mathbf{v}$ by $[\mathbf{v}]$ if indices are not particularly involved. An $(s,t)$-vector sharing scheme involves $s$ participants, inputs a vector $\mathbf{v}$, and outputs to each participant an $(s,t)$-secret sharing of $\mathbf{v}$. A vector sharing scheme is called (information-theoretically) secure, if

$$\Pr(\mathbf{v} = \mathbf{x} | \mathbf{v}_{i_1}, ..., \mathbf{v}_{i_{t-1}}) = \Pr(\mathbf{v} = \mathbf{x}' | \mathbf{v}_{i_1}, ..., \mathbf{v}_{i_{t-1}})$$

for any subset $\{i_1, ..., i_{t-1}\} \subset \{1, ..., s\}$ and any $\mathbf{x}, \mathbf{x}'$ in the ambient space.

**Remark 3.2.** By default, we set $t = s$. In the ideal setting, a vector sharing scheme over finite fields consists of a trusted party and $s$ participants. The trusted party takes the input $\mathbf{v}$, randomly samples vectors $\mathbf{r}_1, ..., \mathbf{r}_{s-1}$ and sends them to the first $s-1$ participants respectively. It then sends $\mathbf{v} - \mathbf{r}_1 - ... - \mathbf{r}_{s-1}$ to the last participant.

### 3.2 Definition of tensor triple

**Definition 3.3** (Tensor triple)**.** By definition, a tensor triple $(\mathbf{u}, \mathbf{v}, W)$ consists of data $\mathbf{u} \in K^m, \mathbf{v} \in K^n, W \in M_{m\times n}(K)$ satisfying $\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^T = W$.

**Definition 3.4** (Tensor triple sharing)**.** Let $P_1, ..., P_s$ 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

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

for the participant $P_i$, such that $\mathbf{u}_i, \mathbf{v}_i$ form secret sharings of $\mathbf{u} \in K^m, \mathbf{v} \in K^n$ respectively, with the relation $\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^T = \sum_{i=1}^{s} W_i$. Shares of $\mathbf{u}, \mathbf{v}$ and $W$ will be denoted by $[\mathbf{u}], [\mathbf{v}]$ and $[W]$ if indices are not particulary involved. For our convenience, such a triple will be called an $(m,n)$-triple. A tensor triple sharing scheme is called secure if the distribution of $\mathbf{u}, \mathbf{v}$ to an adversary who obtains $\{(\mathbf{u}_i, \mathbf{v}_i, W_i)\}_{i \neq i^*}$ for any $i^* \in \{1, ..., s\}$ is computationally indistinguishable from the uniform one.

## 4 Tensor triple generation

### 4.1 RLWE-based two-party tensor triple generation

A batch generation method for Beaver triples based on RLWE-AHE has been introduced in [RSS19]. It can be modified as follows to generate tensor triples.

RLWE-based two-party tensor triple generation $\mathsf{TT.Gen}(m,n,\lambda)$

| **Party** $P_0$ | | **Party** $P_1$ |
|---|---|---|
| $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{AHE.Gen}(1^{\lambda})$ | | |
| | $\xrightarrow{\mathsf{pk}}$ | |
| $\mathbf{u}_0 \leftarrow_{\$} K^m, \mathbf{v}_0 \leftarrow_{\$} K^n$ | | $\mathbf{u}_1 \leftarrow_{\$} K^m, \mathbf{v}_1 \leftarrow_{\$} K^n$ |
| $\mathsf{ct}_{\mathbf{u}} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, \mathbf{u}_0)$ | | $R_{\mathbf{u}} \leftarrow_{\$} M_{m\times n}(K)$ |
| $\mathsf{ct}_{\mathbf{v}} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, \mathbf{v}_0)$ | | $R_{\mathbf{v}} \leftarrow_{\$} M_{n\times m}(K)$ |
| | | $\vec{\mathsf{ct}}_{R_{\mathbf{u}}} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, R_{\mathbf{u}})$ |
| | | $\vec{\mathsf{ct}}_{R_{\mathbf{v}}} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, R_{\mathbf{v}})$ |
| | $\xrightarrow{\mathsf{ct}_{\mathbf{u}}, \mathsf{ct}_{\mathbf{v}}}$ | |
| | | $\vec{\mathsf{ct}}_1 \leftarrow \mathbf{u}_1 \otimes \mathsf{ct}_{\mathbf{v}}$ |
| | | $\vec{\mathsf{ct}}_2 \leftarrow \mathbf{v}_1 \otimes \mathsf{ct}_{\mathbf{u}}$ |
| | | $\vec{\mathsf{ct}}_{W_{\mathbf{u}}} \leftarrow \vec{\mathsf{ct}}_1 + \vec{\mathsf{ct}}_{R_{\mathbf{u}}}$ |
| | | $\vec{\mathsf{ct}}_{W_{\mathbf{v}}} \leftarrow \vec{\mathsf{ct}}_2 + \vec{\mathsf{ct}}_{R_{\mathbf{v}}}$ |
| | $\xleftarrow{\vec{\mathsf{ct}}_{W_{\mathbf{u}}}, \vec{\mathsf{ct}}_{W_{\mathbf{v}}}}$ | |
| $W_{\mathbf{u}} \leftarrow \mathsf{AHE.Dec}(\mathsf{sk}, \vec{\mathsf{ct}}_{W_{\mathbf{u}}})$ | | |
| $W_{\mathbf{v}} \leftarrow \mathsf{AHE.Dec}(\mathsf{sk}, \vec{\mathsf{ct}}_{W_{\mathbf{v}}})$ | | |
| $W_0 \leftarrow W_{\mathbf{u}} + W_{\mathbf{v}}^T + \mathbf{u}_0 \otimes \mathbf{v}_0$ | | $W_1 \leftarrow \mathbf{u}_1 \otimes \mathbf{v}_1 - R_{\mathbf{u}} - R_{\mathbf{v}}^T$ |
| **Output** $\mathbf{u}_0, \mathbf{v}_0, W_0$ | | **Output** $\mathbf{u}_1, \mathbf{v}_1, W_1$ |

## 4.2 sVOLE-based two-party tensor triple generation

An OT-based protocol for generating Beaver multiplication triples was proposed in [Gil99]. In [DSZ15], the authors established a more efficient protocol based on Correlated-OT extension, which has been shown to outperform the AHE based generation method. In this section, we propose an sVOLE-based method to efficiently generate tensor triples.

Observe that under a fixed $\mathbb{F}_p$-vector space isomorphism $\varphi : \mathbb{F}_{p^m} \to \mathbb{F}_p^m$ (for instance, $\varphi(a_0 + a_1\alpha + ... + a_{m-1}\alpha^{m-1}) = (a_0, ..., a_{m-1})$ where $\mathbb{F}_{p^m}$ is realized through an extension $\mathbb{F}_p[\alpha]$ by adding a root of an irreducible monic polynomial of degree $m$), $\mathscr{F}^{p,m,n}_{\text{RsVOLE}}$ becomes the following functionality $\mathscr{F}^{p,m,n}_{\text{RsVOLE}'}$:

Functionality $\mathscr{F}^{p,m,n}_{\text{RsVOLE}'}$

**Players:** The sender $S$ and the receiver $R$.

**Inputs:** None.

**Outputs:**

- $S$ outputs $\mathbf{x} \in_r \mathbb{F}_p^m, V \in_r \mathbb{F}_p^{mn}$;
- $R$ outputs $\mathbf{y} \in_r \mathbb{F}_p^n, U = \mathbf{x} \otimes \mathbf{y} + V \in \mathbb{F}_p^{mn}$.

Given a protocol $\mathsf{RsVOLE}(m,n,\lambda)$ realizing $\mathscr{F}^{p,m,n}_{\text{RsVOLE}'}$, we can easily formulate a way to generate a tensor multiplication triple as follows:

sVOLE-based two-party tensor triple generation $\mathsf{TT.Gen}(m, n, \lambda)$

**Inputs:** None.

**Primitive:** A random subfield VOLE protocol $\mathsf{RsVOLE}(m, n, \lambda)$.

**Interacting Phase:**

1. $P_0$ and $P_1$ launch the protocol $\mathsf{RsVOLE}(m, n, \lambda)$ where $P_0$ acts as the sender and $P_1$ acts as the receiver. $P_0$ obtains $\mathbf{x} \in_r \mathbb{F}_p^m, V \in_r \mathbb{F}_p^{mn}$ and $P_1$ obtains $\mathbf{y} \in_r \mathbb{F}_p^n, U = \mathbf{x} \otimes \mathbf{y} + V \in \mathbb{F}_p^{mn}$;
2. $P_0$ and $P_1$ launch the protocol $\mathsf{RsVOLE}(m, n, \lambda)$ where $P_0$ acts as the receiver and $P_1$ acts as the sender. $P_1$ obtains $\mathbf{x}' \in_r \mathbb{F}_p^m, V' \in_r \mathbb{F}_p^{mn}$ and $P_0$ obtains $\mathbf{y}' \in_r \mathbb{F}_p^n, U' = \mathbf{x}' \otimes \mathbf{y}' + V' \in \mathbb{F}_p^{mn}$;

**Outputs:**

- $P_0$ outputs $(\mathbf{u}_0 = \mathbf{x}, \mathbf{v}_0 = \mathbf{y}', W_0 = -V + U' + \mathbf{x} \otimes \mathbf{y}')$;
- $P_1$ outputs $(\mathbf{u}_1 = \mathbf{x}', \mathbf{v}_1 = \mathbf{y}, W_1 = U - V' + \mathbf{x}' \otimes \mathbf{y})$.

In particular, when $p = 2$, one may also use COT protocols in an analogous way to generate tensor triples over fields with even characteristics.

We use two ways to materialize $\mathsf{RsVOLE}(m, n, \lambda)$. The first method is to use COT for the case when $p = 2$. The idea is similar to the one implemented in [BCG+19], and we simply adapt it in tensor form to suit our need.

COT-based $\mathsf{RsVOLE}(m, n, \lambda)$

| **Receiver** $R$ | | **Sender** $S$ |
|---|---|---|
| **Input** $\mathbf{y} \leftarrow_\$ \{0,1\}^n$ | | **Input** $\mathbf{x} \leftarrow_\$ \{0,1\}^m$ |
| $\mathbf{y} = (y_1, ..., y_n)$ | $\xrightarrow{y_i}$ $\boxed{\text{Random OT}}$ | |
| | $\xleftarrow{\mathsf{seed}_{i,y_i}}$ $\quad$ $\xrightarrow{\mathsf{seed}_{i,0}, \mathsf{seed}_{i,1}}$ | |
| | | $\mathbf{s}_i = \mathsf{PRG}(\mathsf{seed}_{i,0}) \oplus \mathsf{PRG}(\mathsf{seed}_{i,1}) \oplus \mathbf{x}$ |
| | $\xleftarrow{\mathbf{s}_1, ..., \mathbf{s}_n}$ | |
| $\mathbf{u}_i = \mathsf{PRG}(\mathsf{seed}_{i,y_i}) \oplus y_i \mathbf{s}_i$ | | |
| $U = (\mathbf{u}_1^T, ..., \mathbf{u}_n^T)$ | | $V = (\mathsf{PRG}(\mathsf{seed}_{i,0})^T)$ |
| **Output** $\mathbf{y}, U$ | | **Output** $\mathbf{x}, V$ |

The second way is to use Silent OT (SOT). SOT procotol can be directly used to fulfill $\mathsf{RsVOLE}(m, n, \lambda)$. We omit the detail and refer to [BCG+19] for brevity.

## 4.3 Third-party 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 [ST19, MK22]. 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 $(s, t)$ and $(n - s, n - t)$ 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.

## 5 Secure multi-dimensional arithmetic evaluations

In this section we will discuss various vector operations which may be performed securely using Beaver's method. Let us first discuss the degenerated case where $m = n = 1$.

### 5.1 Regular Beaver triple

When $m = n = 1$, we see that a tensor triple is nothin else but a regular Beaver triple. Therefore, we see that arithmetic of numbers may be performed as being well-known. For brevity purposes, we would assume in the remaining sections that the arithmetic of numbers is always securely multi-party computable.

### 5.2 Vector addition and operations involving constants

As expected, additions and operations involving constants may all be computed locally without any interaction by the homomorphicity. More specifically, the following operations are securely multi-party computable and the computation can be done locally without any interaction:

- $[\mathbf{a}] + [\mathbf{b}] = [\mathbf{a} + \mathbf{b}]$;
- Given a public constant vector $\mathbf{c}$, $(\mathbf{a} + \mathbf{c})_1 = \mathbf{a}_1 + \mathbf{c}$, $(\mathbf{a} + \mathbf{c})_i = \mathbf{a}_i$ for $i \neq 1$.
- $c[\mathbf{a}] = [c\mathbf{a}]$, where $c \in K$ is a constant number;
- $\mathbf{c} \cdot [\mathbf{a}] = [\mathbf{c} \cdot \mathbf{a}]$, where $a \in K^n$, and $\mathbf{c} \in K^n$ is a constant vector;
- $\mathbf{c} \otimes [\mathbf{a}] = [\mathbf{c} \otimes \mathbf{a}]$, where $\mathbf{c}$ is a constant vector.

### 5.3 Dot product

Now let us consider the dot product of two vectors $\mathbf{a}, \mathbf{b}$ in the same dimension $n$. Suppose we have a $(n, n)$-triple. First we denote

$$\mathbf{a} - \mathbf{u} = \mathbf{s}, \quad \mathbf{b} - \mathbf{v} = \mathbf{t}.$$

A direct computation gives us

$$\begin{aligned}\mathbf{a} \cdot \mathbf{b} &= (\mathbf{s} + \mathbf{u}) \cdot (\mathbf{t} + \mathbf{v}) \\ &= \mathbf{s} \cdot \mathbf{t} + \mathbf{u} \cdot \mathbf{t} + \mathbf{v} \cdot \mathbf{s} + \mathbf{u} \cdot \mathbf{v} \\ &= \mathbf{s} \cdot \mathbf{t} + \mathbf{u} \cdot \mathbf{t} + \mathbf{v} \cdot \mathbf{s} + \mathrm{tr}(W).\end{aligned}$$

Therefore, by Beaver's method, each party may annouce its share of $\mathbf{s}$ and $\mathbf{t}$ to recover the two vectors. Then they could securely compute the shares as

$$[\mathbf{a} \cdot \mathbf{b}] = \mathbf{s} \cdot \mathbf{t} + [\mathbf{u}] \cdot \mathbf{t} + [\mathbf{v}] \cdot \mathbf{s} + \mathrm{tr}([W]).$$

Note that the last term uses the fact that the trace function is linear.

## 5.4 Outer product

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. Each party $P_i$ secretly computes $(\mathbf{a} \otimes \mathbf{b})_i \leftarrow \mathbf{s} \otimes \mathbf{t} + \mathbf{u}_i \otimes \mathbf{t} + \mathbf{s} \otimes \mathbf{v}_i + W_i$

**Outputs:** $P_i$ outputs $(\mathbf{a} \otimes \mathbf{b})_i$.

One can use exactly the same method to fulfill the need of an outer product. Let us suppose all parties would like to compute the outer product $\mathbf{a} \otimes \mathbf{b}$ of two vectors $\mathbf{a} \in K^m$ and $\mathbf{b} \in K^n$ with the help of an $(m, n)$-triple $(\mathbf{u}, \mathbf{v}, W)$. First we denote

$$\mathbf{a} - \mathbf{u} = \mathbf{s}, \quad \mathbf{b} - \mathbf{v} = \mathbf{t}.$$

Then similarly we could compute

$$\begin{aligned} \mathbf{a} \otimes \mathbf{b} &= (\mathbf{s} + \mathbf{u}) \otimes (\mathbf{t} + \mathbf{v}) \\ &= \mathbf{s} \otimes \mathbf{t} + \mathbf{u} \otimes \mathbf{t} + \mathbf{s} \otimes \mathbf{v} + \mathbf{u} \otimes \mathbf{v} \\ &= \mathbf{s} \otimes \mathbf{t} + \mathbf{u} \otimes \mathbf{t} + \mathbf{s} \otimes \mathbf{v} + W. \end{aligned}$$

Therefore we could similarly make a protocol described above.

One thing to be noted is that the triple is not aimed at fully masking the output matrix $\mathbf{a} \otimes \mathbf{b}$. This output matrix has rank 1 and all entries will be determined once the first row and the first column are determined. As the relation is revealed from the expression itself, using a matrix $W$ of rank 1 to mask the output has exactly the same effect as using a completely random matrix.

## 5.5 Matrix product

As a direct application of the outer product, we could now introduce a way to securely compute a matrix product of two matrices

$$A \in M_{m\times k}(K), \quad B \in M_{k\times n}(K).$$

First denote the columns of $A$ by $A = (\mathbf{a}_1, ..., \mathbf{a}_k)$, and the rows of $B$ by $B = (\mathbf{b}_1, ..., \mathbf{b}_k)^T$. As is well-known, the matrix product has the following outer product expansion formula:

$$AB = \sum_{i=1}^{k} \mathbf{a}_i \otimes \mathbf{b}_i.$$

Therefore, it is easy to formulate a way to compute the matrix product from the outer product primitive. All the parties may simultaneously perform $k$ primitives to compute each summand in the formula above and then individually add the results together without any further interaction. A protocol is described as follows:

Secure Multi-Party Matrix Product $\mathsf{MatProd}(A, B)$

**Input:** $P_i$ inputs shares $A_i, B_i$ of two matrices $A \in M_{m\times k}(K), B \in M_{k\times n}(K)$.

**Primitive:** A tensor triple generation protocol $\mathsf{TT.Gen}(m, n, \lambda)$, secure multi-party outer product protocol $\mathsf{Out}(\mathbf{a}, \mathbf{b})$.

**Pre-processing Phase:** The participants perform $k$ times of a $\mathsf{TT.Gen}(m, n, \lambda)$ protocol. $P_i$ outputs

$$(\mathbf{u}_i^{(j)}, \mathbf{v}_i^{(j)}, W_i^{(j)}) \quad (j = 1, ..., k).$$

**Initial Phase:** Each party $P_i$ pairs columns of $A$ and transposes of rows of $B$ by indices and obtains $k$ pairs of vectors $(\mathbf{a}_l, \mathbf{b}_l), l = 1, ..., k$.

**Interacting Phase:** All parties perform $\mathsf{Out}(\mathbf{a}_l, \mathbf{b}_l)$ for $l = 1, ..., k$. Each $P_i$ obtains $(\mathbf{a}_l \otimes \mathbf{b}_l)_i$.

**Outputs:** $P_i$ outputs $(AB)_i \leftarrow \sum_{l=1}^{k} (\mathbf{a}_l \otimes \mathbf{b}_l)_i$.

## 5.6 Advantages of tensor triple

Secure multi-party matrix multiplication can also be realized using Beaver triples. For a matrix multiplication of size $(m, k)$ by $(k, n)$, we need $k$ of $(m, n)$-triples or $mnk$ Beaver triples. While the cost may not seem to differ much in low dimensions, the cost of Beaver triples will increase quadratically as the size of matrices increases. As an example, to carry out a secure multi-party matrix multiplication between two $1024 \times 1024$ matrices, we need $2^{30}$ Beaver triples in total. Even the generation process of this many Beaver triples is a burden to all the parties. For instance, if we use RLWE-based method [RSS19] for Beaver triple generation, the communication cost reaches an astonishing 256TB. Meanwhile, the parties only need to consume 1024 tensor triples to accomplish the computation. Due to the batch generation nature of sVOLE, it is much easier to generate tensor triples of the required amount.

On the other hand, the tensor multiplication triples are more flexible and applicable for matrix operations. As mentioned in previous sections, any $(n, n)$-triple can be trimmed to serve as two triples of size $(s, t)$ and $(n - s, n - t)$ respectively, for any $s, t < n$. Therefore, secure multi-party matrix operations with different matrix sizes can be achieved using tensor triples of a universal size.

More importantly, when tensor triple technique is applied to achieve the secure multi-party matrix multiplication $A \cdot B$, the number of columns of $A$ (which equals the number of rows of $B$) can be arbitrary. This is extremely convenient in some applications, for example the ones we will introduce in Chapter 6. For instance, if the parties choose to generate "matrix triples" $([X], [Y], [Z] = [X \cdot Y])$ to assist the computation of matrix multiplication, since the size of matrices may not be determined beforehand by the nature of "pre"-computation process, the parties must choose the dimensions of $X, Y, Z$ large enough to satisfy all possible needs, and this may turn out to become an overkill when the protocol is executed. Hence it is convenient to use tensor triples, as the method does not require the parties to know anything about $k$ while still being able to accomplish 100% utilization of the pre-computed triples.

# 6 Applications

## 6.1 Batched squared Euclidean distance computing

Squared Euclidean distance is a widely-used and crucial function in biometric identification and machine learning. In biometric identification, it is often the case the client needs to launch multiple queries, and the server needs to compute the squared Euclidean distance between each query with all references in its own dataset. This type of batched queries essentially portraits a matrix multiplication functionality. Therefore one can use tensor triple to accelerate this process. We will explain by presenting concrete examples as follows.

## 6.2 Batched privacy-preserving biometric identification

In this chapter we show applications for vector triples on privacy-preserving biometric identification protocols. Biometric identification, such as face recognition and fingerprint recognition, often involves computation between biometric samples and references in a dataset. In this scenario, Euclidean distance computing between a vector and a fixed family of vectors is implemented each time a query is launched. We demonstrate tensor triples can be used for batched queries in privacy-preserving biometric identification protocols to extremely increase the efficiency.

### 6.2.1 FingerCode

FingerCode [JPHP99, HMEK11] is a fingerprint recognition algorithm. In the setting of FingerCode, the server holds a dataset of references $Y = (\mathbf{y}_1^T, ..., \mathbf{y}_n^T) \in M_{k\times n}(K)$. The client would like to securely make a batch of queries $X = (\mathbf{x}_1^T, ..., \mathbf{x}_m^T)^T \in M_{m\times k}(K)$ for recognition. The protocol should proceed as described below.

1. The parties obtain shares of the squared Euclidean distance between the queries and references from the dataset. In expressions, the parties securely compute $D = D_X + D_Y - XY \in M_{m\times n}(K)$, where $D_X = (\mathbf{x}_1\mathbf{x}_1^T, ..., \mathbf{x}_m\mathbf{x}_m^T) \otimes (1, 1, ..., 1)$ and $D_Y = (1, 1, ..., 1) \otimes (\mathbf{y}_1\mathbf{y}_1^T, ..., \mathbf{y}_n\mathbf{y}_n^T)$;

2. The parties securely compare entries of $D$ with the predetermined threshold $d$. Recognize $\mathbf{x}_i$ as $\mathbf{y}_j$ if $D_{ij} \le d$;

Note $D_X$ and $D_Y$ can be computed locally. Therefore the first step can be fulfilled by implementing $\mathsf{MatProd}(X, Y)$ using tensor triples. The second step can be done using any regular implementation of the secure comparison protocol.

### 6.2.2 Eigenfaces

Eigenfaces [TP91] is a classical face recognition algorithm. In the setting of Eigenfaces, the server holds a dataset of Eigenfaces $U = (\mathbf{u}_1^T, ..., \mathbf{u}_n^T) \in M_{k\times n}(K)$, an average face $\bar{\mathbf{u}} \in K^k$, and a dataset of $N$ projected faces $Y = (\mathbf{y}_1^T, ..., \mathbf{y}_N^T) \in M_{n\times N}(K)$. The client would like to securely make a batch of queries $X = (\mathbf{x}_1^T, ..., \mathbf{x}_m^T)^T \in M_{m\times k}(K)$ for recognition. The protocol should proceed as described below.

1. The parties subtract the average face and obtain shares of the projected faces $\bar{X} = (\bar{\mathbf{x}}_1^T, ..., \bar{\mathbf{x}}_m^T)^T$ onto the eigenbasis. More specifically, the parties securely compute $\bar{X} = (X - \bar{U})U \in M_{m\times n}(K)$, where $\bar{U} = (\bar{\mathbf{u}}^T, ..., \bar{\mathbf{u}}^T)^T \in M_{m\times k}(K)$;

2. The parties obtain shares of the squared Euclidean distance between the projected faces and those in the dataset. In expressions, the parties securely compute $D = D_X + D_Y - 2\bar{X}Y \in M_{m\times N}(K)$, where $D_X = (\bar{\mathbf{x}}_1\bar{\mathbf{x}}_1^T, ..., \bar{\mathbf{x}}_m\bar{\mathbf{x}}_m^T) \otimes (1, 1, ..., 1)$ and $D_Y = (1, 1, ..., 1) \otimes (\mathbf{y}_1\mathbf{y}_1^T, ..., \mathbf{y}_N\mathbf{y}_N^T)$. More specifically, they use Beaver triple to compute $\langle \bar{\mathbf{x}}_i, \bar{\mathbf{x}}_i \rangle$ and tensor triple to compute $\bar{X}Y$. Also note $D_Y$ can be computed locally;

3. The parties securely compare entries of $D$ with the predetermined threshold $d$. Recognize $\mathbf{x}_i$ as $\mathbf{y}_j$ if $D_{ij} \le d$;

### 6.2.3 FaceNet

FaceNet [SKP15] is a more recent facial recognition system based on deep learning. It was proposed in 2015 and successfully provided a way to generate a very high-quality mapping from face images to their vector representatives. In its setting, the server holds a dataset of references $Y = (\mathbf{y}_1^T, ..., \mathbf{y}_n^T) \in M_{k\times n}(K)$. The client would like to securely make a batch of queries $X = (\mathbf{x}_1^T, ..., \mathbf{x}_m^T)^T \in M_{m\times k}(K)$ for recognition. We assume all the data have been pre-processed through a well-trained network. The protocol should proceed as described below.

1. The parties obtain shares of the squared Euclidean distance between the queries and references from the dataset. In expressions, the parties securely compute $D = D_X + D_Y - XY \in M_{m\times n}(K)$, where $D_X = (\mathbf{x}_1\mathbf{x}_1^T, ..., \mathbf{x}_m\mathbf{x}_m^T) \otimes (1, 1, ..., 1)$ and $D_Y = (1, 1, ..., 1) \otimes (\mathbf{y}_1\mathbf{y}_1^T, ..., \mathbf{y}_n\mathbf{y}_n^T)$;

2. The parties securely compare entries of $D$ with the predetermined threshold $d$. Recognize $\mathbf{x}_i$ as $\mathbf{y}_j$ if $D_{ij} \le d$;

Here again, $D_X$ and $D_Y$ can be computed locally. The MPC part of the overall protocol proceeds exactly the same as in the FingerCode case.

As a remark, we see that the flexibility of tensor triples allows us to apply all protocols above on a dataset of vectors in an arbitrary dimension. This is extremely useful as dimensions of data points many vary in different settings. Therefore, tensor triple generation may well be considered as a genuine "pre-computation" process, as the triples generated are suitable for MPC on datasets in all dimensions.

# 7 Implementation and performance

In this chapter, our implementations are based on C++. The experiments are run on desktop with AMD 3950X CPU and 48GB RAM. We considered both simulated LAN and WAN environments with 500 mbps bandwidth and 20 ms one-way latency. The protocols are suitable for multi-threading by parallel computation, but we measure the performance in single thread setting. The experiments are executed multiple times and the medians of the results are presented in the following tables.

## 7.1 Tensor triple generation

We mainly choose to use the subfield VOLE method to generate triples as it generally have a better performance than the RLWE method. We implement both Correlated OT based protocol and silent OT based protocol for RsVOLE. We also use libOTe library to fulfill basic functionalities such as COT and OT extension. For silent OT, we used the expand-convolute code from [RRT23] with expander weight 7 and convolution state size 24. The hamming weight of the sparse vector in this setting is 400, which means each silent VOLE requires an execution of an OT extension based subfield VOLE of size 400.

It can be seen from the tables that COT method is more efficient when tensor triples of small sizes are needed, while SOT method allows generation of tensor triples of large sizes with moderate communication cost.

**Table 1.** Performance of COT-based 32-bit $(m, n)$-tensor triple generation (in milliseconds). For each size we generate $1, 2^5, 2^{10}$ number of triples (arranged in rows) and run the protocol both in LAN and WAN environments (arranged in columns).

| $n$ $m$ | $2^3$ | | $2^8$ | | $2^{10}$ | | $2^{14}$ | |
|---|---|---|---|---|---|---|---|---|
| $2^3$ | 10 | 188 | 10 | 312 | 12 | 394 | 53 | 985 |
| | 14 | 317 | 23 | 649 | 60 | 1596 | 748 | 19153 |
| | 287 | 5836 | 567 | 15354 | 1771 | 45577 | 22378 | 597527 |
| $2^8$ | | | 29 | 654 | 108 | 1603 | 3906 | 22553 |
| | | | 409 | 9356 | 1737 | 37379 | 85391 | 602922 |
| | | | 12603 | 309499 | 57274 | 1181690 | | |
| $2^{10}$ | | | | | 638 | 5530 | 22323 | 92565 |
| | | | | | 14003 | 149363 | 567917 | 2665457 |
| | | | | | 425520 | 4732780 | | |

**Table 2.** Performance of SOT-based 32-bit $(m, n)$-tensor triple generation (in milliseconds). For each size we generate $1, 2^5, 2^{10}$ number of triples (arranged in rows) and run the protocol both in LAN and WAN environments (arranged in columns).

| $n$ $m$ | $2^3$ | | $2^8$ | | $2^{10}$ | | $2^{14}$ | |
|---|---|---|---|---|---|---|---|---|
| $2^3$ | 104 | 529 | 93 | 528 | 93 | 576 | 193 | 691 |
| | 3865 | 4554 | 3765 | 4651 | 3840 | 4690 | 7922 | 7972 |
| | 123127 | 143559 | 122763 | 143203 | 122937 | 140101 | 257346 | 267526 |
| $2^8$ | | | 593 | 1603 | 591 | 1646 | 763 | 1733 |
| | | | 24532 | 33195 | 24534 | 32774 | 31096 | 37629 |
| | | | 786790 | 1047499 | 788486 | 1052955 | 1006983 | 1193853 |
| $2^{10}$ | | | | | 2271 | 4850 | 2647 | 5252 |
| | | | | | 92154 | 137287 | 104976 | 142970 |
| | | | | | 2991083 | 4441299 | 3395699 | 4606848 |

**Table 3.** Communication cost of COT-based 32-bit $(m, n)$-tensor triple generation (in megabytes). For each size we test the cost for $1, 2^5, 2^{10}$ number of triples (arranged in rows).

| $n$ $m$ | $2^3$ | $2^8$ | $2^{10}$ | $2^{14}$ |
|---|---|---|---|---|
| $2^3$ | 0.032 | 0.52 | 2.02 | 32.02 |
| | 0.76 | 16.26 | 64.26 | 1024.26 |
| | 12.10 | 260.10 | 1028.10 | 16388.08 |
| $2^8$ | | 16.26 | 64.26 | 1024.26 |
| | | 520.01 | 2056.01 | 32776.01 |
| | | 8320.08 | 32896.08 | |
| $2^{10}$ | | | 257.01 | 4097.02 |
| | | | 8224.01 | 131104.04 |
| | | | 130969.60 | |

**Table 4.** Communication cost of SOT-based 32-bit $(m, n)$-tensor triple generation (in megabytes). For each size we test the cost for $1, 2^5, 2^{10}$ number of triples (arranged in rows).

| $m$ \ $n$ | $2^3$ | $2^8$ | $2^{10}$ | $2^{14}$ |
|---|---|---|---|---|
| $2^3$ | 1.00 | 1.00 | 1.00 | 1.23 |
| | 32.86 | 32.86 | 32.86 | 39.31 |
| | 525.82 | 525.82 | 525.82 | 628.95 |
| $2^8$ | | 28.79 | 28.79 | 28.99 |
| | | 921.21 | 921.21 | 927.65 |
| | | 14739.36 | 14739.36 | 14842.48 |
| $2^{10}$ | | | 114.76 | 114.96 |
| | | | 3672.21 | 3678.65 |
| | | | 58755.36 | 58858.48 |

We present here also a high-level analysis of the two methods. In small cases, COT-based generation method is straightforward and faster, while it may take a considerable amount of time for SOT-based method to finish the structure building for the protocol. When one deals with matrices of large dimensions or needs a large amount of tensor triples, since there is a linear overhead in the communication cost of COT-based generation method, the data transfer may become intolerable for the parties to generate these triples. While on the other hand, as the communication cost of the SOT-based generation method is sublinear asymptotically, it requires much less communication to generate all the necessary tensor triples.

As a brief comparison, based on the performance tables provided by [RSS19], it takes approximately 2300 milliseconds to generate one $(2^{10}, 2^{10})$-triple RLWE-based tensor triple generation method, with a communication cost of 256MB. We see from the example that VOLE-based generation method indeed performs better.

## 7.2 Matrix multiplication

For $l$-bit matrix multiplication of size $(m, k)$ by $(k, n)$, we need $k$ pairs vector triples of size $(m, n)$ for $k$ outer products. The total online communication is therefore is $2lk(m + n)$ bits. As a reference, the total communication cost for applying Beaver triple to perform matrix multiplication is $2lkmn$ bits. We see the usage of tensor triples accelerates the computation significantly. In the following tables, we choose $l = 32$.

**Table 5.** Performance of tensor triple-based 32-bit $(m, k) \times (k, n)$ matrix multiplication time (in milliseconds). The protocol is run both in LAN and WAN environments (arranged in columns).

| $k$ \ $(m, n)$ | $(2^3, 2^3)$ | | $(2^5, 2^5)$ | | $(2^8, 2^8)$ | | $(2^{10}, 2^{10})$ | |
|---|---|---|---|---|---|---|---|---|
| $2^3$ | 1 | 26 | 1 | 26 | 1 | 27 | 11 | 44 |
| $2^8$ | 1 | 26 | 1 | 37 | 22 | 62 | 292 | 323 |
| $2^{10}$ | 1 | 37 | 2 | 44 | 71 | 128 | 1062 | 1289 |

**Table 6.** Communication cost of tensor triple-based 32-bit $(m, k) \times (k, n)$ matrix multiplication (in megabytes).

| $k$ \ $(m, n)$ | $(2^3, 2^3)$ | $(2^5, 2^5)$ | $(2^8, 2^8)$ | $(2^{10}, 2^{10})$ |
|---|---|---|---|---|
| $2^3$ | 0.001 | 0.004 | 0.031 | 0.125 |
| $2^8$ | 0.031 | 0.125 | 1 | 4 |
| $2^{10}$ | 0.125 | 0.5 | 4 | 16 |

## 7.3 Batched privacy-preserving biometric identification

In this section we present tensor triple based implementations of batched queries of FingerCode [JPHP99] and Eigenfaces [TP91] protocols with a comparison of the efficiency with the GSHADE [BCF+14] protocol, and FaceNet [SKP15] with a comparison with the [NB18] protocol. For FingerCode, we use 640-dimensional vectors of 8-bit elements, and we use 32 bits to record each square Euclidean distance. For EigenFaces, we use 10304-dimensional vectors of 8-bit elements, and we use 32 bits to record each square Euclidean distance. For FaceNet, the database consists of 128-dimensional vectors of elements with floating point precision, but a truncation will be applied to all of the elements to map them into 8-bit strings, and each final square Euclidean distance consumes 64 bits. It can be seen from the comparison the tensor triple significantly accelerates the identification process. The data for the performance of FingerCode and FaceNet protocols are collected individually according to our experiments. The experiments for all implementations are run in the same environment introduced at the beginning of this chapter.

**Table 7.** Performance of secure squared Euclidean distance computation in batched FingerCode protocol (128 queries)

| | $N = 128$ | | $N = 1024$ | |
|---|---|---|---|---|
| Protocol | [BCF+14] | Ours | [BCF+14] | Ours |
| Time (s) | 154.37 | **0.016** | 176.64 | **0.082** |
| Communication (MB, online) | 1688.71 | **1.25** | 5379.24 | **5.63** |

Clearly, tensor triple method achieves a much better online performance according to the comparison.

**Table 8.** Performance of Eigenfaces protocol without the secure comparison step (80 queries)

| | $N = 320$ | $N = 1000$ |
|---|---|---|
| Protocol | Ours | Ours |
| Time (s) | 0.029 | 0.032 |
| Communication (MB, online) | 14.57 | 14.69 |

As a comparison to the performance in [BCF+14], a single query for the $N = 320$ case would take 0.6 seconds to fulfill, and the corresponding communication cost is 7.7MB. When $N = 1000$, a single query takes 1.6 seconds and costs 9.4MB. Although the performance in [BCF+14] takes also the secure comparison step into consideration, it can still clearly be seen that the tensor triple method behaves much better for batched queries.

**Table 9.** Performance (time in seconds) of secure squared Euclidean distance computation in batched FaceNet protocol for $m$ queries against a database of $n$ references

| Protocol / $(m, n)$ | [NB18] | Ours |
|---|---|---|
| $(2^4, 2^4)$ | 2.58 | **0.00068** |
| $(2^4, 2^{10})$ | 165.95 | **0.0055** |
| $(2^{10}, 2^{10})$ | 10559.68 | **0.25** |

One may argue that there is a pre-computation cost for the tensor triple method. We shall elaborate here with two points. First, even if one considers the generation step, the tensor triple method still performs faster under almost all circumstances. As an instance, it takes only several seconds to generation the necessary amount of tensor triples with correct dimensions for the $N = 128$ case in FingerCode (Check Section 7.1 for details). Second, as we have pointed out, the tensor triple method truly enables the possibility of pre-computation process in secure multi-party matrix computation. Compared to other protocols, for instance GSHADE for the FingerCode protocol, we do not see a way any step in the GSHADE method can be pre-processed without the knowledge of dimensions of the matrices involved. Therefore, it should be considered fair for such comparisons in the tables we listed.

## 8 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.

## References

[ADI+17] B. Applebaum, I. Damgrard, Y. Ishai, M. Nielsen, and L. Zichron, *Secure arithmetic computation with constant computational overhead*, Advances in cryptology – crypto 2017, 2017, pp. 223–254.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, *More efficient oblivious transfer and extensions for faster secure computation*, Proceedings of the 2013 acm sigsac conference on computer & communications security, 2013, pp. 535–548.

[BCF+14] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner, *Gshade: Faster privacy-preserving distance computation and biometric identification*, Proceedings of the 2nd acm workshop on information hiding and multimedia security, 2014, pp. 187–198.

[BCGI18] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, *Compressing vector ole*, Proceedings of the 2018 acm sigsac conference on computer and communications security, 2018, pp. 896–912.

[BCG+19] 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*, Proceedings of the 2019 acm sigsac conference on computer and communications security, 2019, pp. 291–308.

[BCP13a] J. Bringer, H. Chabanne, and A. Patey, *Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends*, 2013, pp. 42–52.

[BCP13b] J. Bringer, H. Chabanne, and A. Patey, *Shade: Secure hamming distance computation from oblivious transfer*, Financial cryptography and data security, 2013, pp. 164–176.

[BEOMC16] J. Bringer, O. El Omri, C. Morel, and H. Chabanne, *Boosting gshade capabilities: New applications and security in malicious setting*, Proceedings of the 21st acm on symposium on access control models and technologies, 2016, pp. 203–214.

[BFCP12] J. Bringer, M. Favre, H. Chabanne, and A. Patey, *Faster secure computation for biometric identification using filtering*, 2012, pp. 257–264.

[BG11] M. Blanton and P. Gasti, *Secure and efficient protocols for iris and fingerprint identification*, Computer security – esorics 2011, 2011, pp. 190–209.

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, *(leveled) fully homomorphic encryption without bootstrapping*, Proceedings of the 3rd innovations in theoretical computer science conference, 2012, pp. 309–325.

[BIK+17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, *Practical secure aggregation for privacy-preserving machine learning*, Proceedings of the 2017 acm sigsac conference on computer and communications security, 2017, pp. 1175–1191.

[CRR21] G. Couteau, P. Rindal, and S. Raghuraman, *Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes*, 2021, pp. 502–534.

[DCH+16] S. Deshmukh, H. Carter, G. Hernandez, P. Traynor, and K. Butler, *Efficient and secure template blinding for biometric authentication*, 2016 ieee conference on communications and network security (cns), 2016, pp. 480–488.

[DGK07] I. Damg**r**ard, M. Geisler, and M. Krøigaard, *Efficient and secure comparison for on-line auctions*, Proceedings of the 12th australasian conference on information security and privacy, 2007, pp. 416–430.

[DJ01] I. Damg**r**ard and M. Jurik, *A generalisation, a simplification and some applications of paillier's probabilistic public-key system*, Public key cryptography, 2001, pp. 119–136.

[DSZ15] D. Demmler, T. Schneider, and M. Zohner, *Aby - a framework for efficient mixed-protocol secure two-party computation*, 2015.

[EFG+09] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, *Privacy-preserving face recognition*, Privacy enhancing technologies, 2009, pp. 235–253.

[FV12] J. Fan and F. Vercauteren, *Somewhat practical fully homomorphic encryption*, IACR Cryptol. ePrint Arch. **2012** (2012), 144.

[GBFG+16] M. Gomez-Barrero, J. Fierrez, J. Galbally, E. Maiorana, and P. Campisi, *Implementation of fixed-length template protection based on homomorphic encryption with application to signature biometrics*, 2016 ieee conference on computer vision and pattern recognition workshops (cvprw), 2016, pp. 259–266.

[GBGMF17] M. Gomez-Barrero, J. Galbally, A. Morales, and J. Fierrez, *Privacy-preserving comparison of variable-length data with application to biometric template protection*, IEEE Access **5** (2017), 8606–8619.

[Gil99] N. Gilboa, *Two party rsa key generation*, Advances in cryptology — crypto' 99, 1999, pp. 116–129.

[HH16] C. Hahn and J. Hur, *Efficient and privacy-preserving biometric identification in cloud*, ICT Express **2** (2016), no. 3, 135–139. Special Issue on ICT Convergence in the Internet of Things (IoT).

[HMEK11] Y. Huang, L. Malka, D. Evans, and J. Katz, *Efficient privacy-preserving biometric identification*, Network and Distributed System Security Symposium (2011).

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, *Extending oblivious transfers efficiently*, Advances in cryptology - crypto 2003, 2003, pp. 145–161.

[JPHP99] A. K. Jain, S. Prabhakar, L. Hong, and S. Pankanti, *Fingercode: a filterbank for fingerprint representation and matching*, Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149) **2** (1999), 187–193 Vol. 2.

[KK13] V. Kolesnikov and R. Kumaresan, *Improved ot extension for transferring short secrets*, Advances in cryptology – crypto 2013, 2013, pp. 54–70.

[KKRT16] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, *Efficient batched oblivious prf with applications to private set intersection*, 2016.

[KPPS21] N. Koti, M. Pancholi, A. Patra, and A. Suresh, *SWIFT: Super-fast and robust Privacy-Preserving machine learning*, 30th usenix security symposium (usenix security 21), August 2021, pp. 2651–2668.

[LCP+12] Y. Luo, S.-c. Cheung, T. Pignata, R. Lazzeretti, and M. Barni, *An efficient protocol for private iris-code matching by means of garbled circuits*, 2012, pp. 2653–2656.

[LPS12] R. D. Labati, V. Piuri, and F. Scotti, *Biometric privacy protection: Guidelines and technologies*, E-business and telecommunications, 2012, pp. 3–19.

[MK22] P. Muth and S. Katzenbeisser, *Assisted mpc*, 2022. `https://eprint.iacr.org/2022/1453`.

[MLL+19] Z. Ma, Y. Liu, X. Liu, J. Ma, and K. Ren, *Lightweight privacy-preserving ensemble classification for face recognition*, IEEE Internet of Things Journal **6** (2019), no. 3, 5778–5790.

[MR18] P. Mohassel and P. Rindal, *Aby3: A mixed protocol framework for machine learning*, Proceedings of the 2018 acm sigsac conference on computer and communications security, 2018, pp. 35–52.

[MRT20] P. Mohassel, M. Rosulek, and N. Trieu, *Practical privacy-preserving k-means clustering*, Proceedings on Privacy Enhancing Technologies **2020** (2020), 414–433.

[NB18] V. Naresh Boddeti, *Secure face matching using fully homomorphic encryption*, 2018 ieee 9th international conference on biometrics theory, applications and systems (btas), 2018, pp. 1–10.

[NP06] M. Naor and B. Pinkas, *Oblivious polynomial evaluation*, SIAM Journal on Computing **35** (2006), no. 5, 1254–1281, available at https://doi.org/10.1137/S0097539704383633.

[NWI$^+$13] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, *Privacy-preserving ridge regression on hundreds of millions of records*, 2013, pp. 334–348.

[OPJM10] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich, *Scifi - a system for secure face identification*, 2010, pp. 239–254.

[Pai99] P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, Advances in cryptology — eurocrypt '99, 1999, pp. 223–238.

[Rab81] M. O. Rabin, *How to exchange secrets with oblivious transfer*, TR-81 edition (1981).

[RRT23] S. Raghuraman, P. Rindal, and T. Tanguy, *Expand-convolute codes for pseudorandom correlation generators from lpn*, Advances in cryptology – crypto 2023, 2023, pp. 602–632.

[RSS19] D. Rathee, T. Schneider, and K. K. Shukla, *Improved multiplication triple generation over rings via rlwe-based ahe*, Cryptology and network security, 2019, pp. 347–359.

[SGRR19] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, *Distributed vector-ole: Improved constructions and implementation*, Proceedings of the 2019 acm sigsac conference on computer and communications security, 2019, pp. 1055–1072.

[SKP15] F. Schroff, D. Kalenichenko, and J. Philbin, *Facenet: A unified embedding for face recognition and clustering*, 2015 ieee conference on computer vision and pattern recognition (cvpr), 2015, pp. 815–823.

[SSNO12] S. F. Shahandashti, R. Safavi-Naini, and P. Ogunbona, *Private fingerprint matching*, Information security and privacy, 2012, pp. 426–433.

[SSW10] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, *Efficient privacy-preserving face recognition*, Information, security and cryptology – icisc 2009, 2010, pp. 229–244.

[ST19] 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).

[TP91] M. Turk and A. Pentland, *Eigenfaces for Recognition*, Journal of Cognitive Neuroscience **3** (1991), no. 1, 71–86, available at https://direct.mit.edu/jocn/article-pdf/3/1/71/1932018/jocn.1991.3.1.71.pdf.

[WK12] K.-S. Wong and M.-H. Kim, *A privacy-preserving biometric matching protocol for iris codes verification*, 2012 third ftra international conference on mobile, ubiquitous, and intelligent computing, 2012, pp. 120–125.

[Yao86] A. C.-C. Yao, *How to generate and exchange secrets*, 27th annual symposium on foundations of computer science (sfcs 1986), 1986, pp. 162–167.

[YWL$^+$20] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, *Ferret: Fast extension for correlated ot with small communication*, Proceedings of the 2020 acm sigsac conference on computer and communications security, 2020, pp. 1607–1626.