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, Beijing 101408, China

Abstract. Over years of the development of secure multi-party computation (MPC), many sophisticated functionalities have been made practical 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, with reasonable offline costs.

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 [46], 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 [17,4,22,42,10,9,21,20,28], privacy-preserving machine learning [35,5,29,30,27] and many other practical fields that require an intensive use of vector and matrix operations, such as vector tensoring and matrix multiplication. Unlike the straightforward 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. For instance, SPDZ framework has been used to boost multi-dimensional MPC protocols [11].

Meanwhile, apart from the classical correlation generation protocols such as 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 [33,2,6,7]. 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 Our 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 [24], Eigenfaces [44] and FaceNet [41]. 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. Our implementation results show that the squared Euclidean distance computation in 128 queries of batched Fingercodes against a database of 1024 only takes 0.082 seconds while the GSHADE takes 176.64 seconds, which is more than 2000 times faster than previous work. We also show that the squared Euclidean distance computation in 80 queries of batched Eigenface against a database of 1000 only takes 0.032 seconds while the GSHADE takes 104 seconds, which is more than 3000 times faster than previous work.

1.3 Related Works

Abspoel et al. in [1] proposed the concept of "outer product triple" through linear secret sharing schemes over Galois rings. The authors have also explained the outer product triples are useful as they can be utilized to assist the secure multi-party computation of outer products between vectors. The definition of outer product triple is equivalent to the notion of tensor triple we will define in Sec. 3, but we will introduce protocols involving these triples for much broader applications. For the parameter setting, we do not require the dimensions of tensor triples equal to the dimensions of target vectors in computation, since the triple generation process is in the preprocessing phase, and making the knowledge of the dimensions of multi-dimensional objects occurred in MPC protocols beforehand is unreasonable. Hence in our definition, the tensor triples can always be generated without any knowledge of the dimensions in advance. For the generation process, we focus on the two-party case and only put interests on the (s, s)-additive secret sharing. We propose two novel and efficient methods based on RLWE and VOLE respectively to generate tensor triples and show in experiments that these methods outperform all current ways to generate tensor triples. For applications, we not only propose the secure multi-party computation of vector outer products, but more functionalities such as matrix multiplication, which is a much more useful functionality in practice. As we introduced in Sec. 1.1, in privacy-preserving biometric identification and privacy-preserving machine learning, the secure multi-party computation of matrix multiplication is an indispensable ingredient of all kinds of protocols. We then further proceed to practical usage of tensor triples by testing the efficiency of the corresponding protocols in biometric identification.

As pointed out, the secure multi-party computation of matrix multiplication is a crucial functionality in many areas. As we are aware, there exist a various of works that attempt to fulfill this functionality more efficiently. The most seemingly practical way is the realization of secure matrix computation through the SPDZ framework (for instance [11,31]). On the other hand, our work only uses comparatively lightweight MPC components such as VOLE to fulfill the functionality. As a result, we will show in detailed statistics that our method obtains a very high efficiency and outperforms the homomorphic encryption based protocols by a large factor.

Similarly, there are multiple works aimed to accelerate the privacy-preserving biometric identification process. For FingerCode, Eigenfaces and FaceNet, the three protocols discussed in this paper, we are also aware of the existence of various works and improvements. Nevertheless, we have not discovered any work using VOLE or similar techniques to improve the efficiency. We will still discuss them in the corresponding sections and show a detailed comparison between different implementations.

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.

1.5 Organization of Paper

In Sec. 2, we define necessary notions and primitives in MPC. In Sec. 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 proceed to introducing several ways to generate tensor triples. The protocols are focused on resolving the generation of tensor triples for two parties, as is the usual need in most practical applications. In Sec. 4 we explain the usage of tensor triples to accomplish MPC of basic algebraic operations for scalars, vectors and matrices. Matrix operations. In Sec. 5, we discuss possible applications of tensor triples in practice. 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 section are merely few instances. In Sec. 6 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.

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 [37], 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 $1 \leq i \leq n$. 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 [23,25,26]) which generates n message pairs of bit-length l is denoted by ROT_l^n. A correlated OT (COT) [3,7,45] is a variant of ROT that allows the sender to pre-determine 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. A subfield vector oblivious linear evaluation (sVOLE) 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 below in Figure 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_m^n

Players: The sender S and the receiver R.
Inputs: $\perp$.
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 $\mathbf{u} \in \mathbb{F}_p^n, x\mathbf{u} + \mathbf{v} \in \mathbb{F}_{p^m}^n$.

Fig. 1. Functionality of RCOT and RsVOLE

VOLE protocols with semi-honest and computational security in OT-hybrid model have been defined in [2,7]. Subfield VOLE, as an important variant of VOLE, has also been implemented in various ways (See [6,7,40,45,12,38]).

2.2 RLWE-Based Additively Homomorphic Encryption (RLWE-AHE)

The notion of an AHE scheme we follow is basically the same as the one in [39], with an addition of a tensorial scalar operation which is needed for further use. Roughly speaking, it is a scheme with "linearity". For brevity, the detailed definition will be introduced in Appendix B.

Instances of an AHE scheme which are not based on RLWE are [36,13,15]. While most RLWE-based instantiations of such a scheme (such as [8,18]) 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

3.1 Definition of Tensor Triple

In this section, we first define the notion of vector sharing.

Definition 1. *An (s,t)-secret sharing of a vector $\boldsymbol{v}$ is a set $\{[\boldsymbol{v}]_1, ..., [\boldsymbol{v}]_s\}$ of s vectors, such that there exists an efficient algorithm to reconstruct $\boldsymbol{v}$ from any t shares $[\boldsymbol{v}]_{i_1}, ..., [\boldsymbol{v}]_{i_t}$, but there is no algorithm that can efficiently reconstruct $\boldsymbol{v}$ from any subset of shares with fewer cardinality. We will denote a share of $\boldsymbol{v}$ by $[\boldsymbol{v}]$ if indices are not particularly involved. The secret sharing is called linear if $[\boldsymbol{u}]_i + [\boldsymbol{v}]_i = [\boldsymbol{u} + \boldsymbol{v}]_i$. A linear (s,t)-vector sharing scheme involves s participants, inputs a vector $\boldsymbol{v}$, and outputs to each participant a linear (s,t)-secret sharing of $\boldsymbol{v}$. A vector sharing scheme is called (information-theoretically) secure, if*

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

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

Remark 2. *In the rest of the paper, we use $t = s$, and 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 $\boldsymbol{v}$, randomly samples vectors $\boldsymbol{r}_1, ..., \boldsymbol{r}_{s-1}$ and sends them to the first $s-1$ participants respectively. It then sends $\boldsymbol{v} - \boldsymbol{r}_1 - ... - \boldsymbol{r}_{s-1}$ to the last participant. The secret vector $\boldsymbol{v}$ is therefore only recoverable by all parties together.*

The notion of tensor triple has been proposed in various forms (See [1]) to more efficiently fulfill MPC of vectors. We propose the following definition to better facilitate its properties.

Definition 3 (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 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 $[\boldsymbol{u}]_i \in K^m, [\boldsymbol{v}]_i \in K^n, [W]_i \in M_{m \times n}(K)$ 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 $\boldsymbol{u} \otimes \boldsymbol{v} = \boldsymbol{u}\boldsymbol{v}^T = \sum_{i=1}^{s} [W]_i$. Shares of $\boldsymbol{u}, \boldsymbol{v}$ and W will be denoted by $[\boldsymbol{u}], [\boldsymbol{v}]$ and $[W]$ if indices are not particularly 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 $\boldsymbol{u}, \boldsymbol{v}$ to an adversary who obtains $\{([\boldsymbol{u}]_i, [\boldsymbol{v}]_i, [W]_i)\}_{i \neq i^*}$ for any $i^* \in \{1, ..., s\}$ is computationally indistinguishable from the uniform one.*

3.2 Tensor Triple Generation

Now we would like to introduce multiple ways to efficiently generate tensor triples. Figure 4 shows the ideal functionality of tensor triple generation. In practice, the generation process can be fulfilled via Ring-LWE or subfield VOLE. We provide below these protocols in details.

RLWE-Based Two-Party Tensor Triple Generation A batch generation method for Beaver triples based on RLWE-AHE has been introduced in [39]. It can be modified as dipicted in Figure 2 to generate tensor triples.

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

Setup: P_0 generates $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{AHE.Gen}(1^\lambda)$ and publish pk.
Interacting Phase:

1. P_0 samples $\mathbf{u}_0 \leftarrow_\$ K^m, \mathbf{v}_0 \leftarrow_\$ K^n$ and encrypts them to obtain the ciphertexts
$$\mathsf{ct}_\mathbf{u} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, \mathbf{u}_0), \mathsf{ct}_\mathbf{v} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, \mathbf{v}_0).$$
P_1 samples $\mathbf{u}_1, \leftarrow_\$ K^m, \mathbf{v}_1 \leftarrow_\$ K^n, R_\mathbf{u} \leftarrow_\$ M_{m\times n}(K), R_\mathbf{v} \leftarrow_\$ M_{n\times m}(K)$ and encrypts them to obtain the ciphertexts $\mathbf{ct}_{R_\mathbf{u}} = \mathsf{AHE.Enc}(\mathsf{pk}, R_\mathbf{u}), \mathbf{ct}_{R_\mathbf{v}} = \mathsf{AHE.Enc}(\mathsf{pk}, R_\mathbf{v})$.
2. P_0 sends to P_1 the ciphertexts $\mathsf{ct}_\mathbf{u}, \mathsf{ct}_\mathbf{v}$;
3. By homomorphicity, P_1 computes $\mathbf{ct}_1 = \mathbf{u}_1 \otimes \mathsf{ct}_\mathbf{v}, \mathbf{ct}_2 = \mathbf{v}_1 \otimes \mathsf{ct}_\mathbf{u}$ and $\mathbf{ct}_{W_\mathbf{u}} = \mathbf{ct}_1 + \mathbf{ct}_{R_\mathbf{u}}, \mathbf{ct}_{W_\mathbf{v}} = \mathbf{ct}_2 + \mathbf{ct}_{R_\mathbf{v}}$ and then sends to P_1 the ciphertexts $\mathbf{ct}_{W_\mathbf{u}}, \mathbf{ct}_{W_\mathbf{v}}$;
4. P_0 performs decryption to obtain $W_\mathbf{u} = \mathsf{AHE.Dec}(\mathsf{sk}, \mathbf{ct}_{W_\mathbf{u}}), W_\mathbf{v} = \mathsf{AHE.Dec}(\mathsf{sk}, \mathbf{ct}_{W_\mathbf{v}})$ and then computes $W_0 = W_\mathbf{u} + W_\mathbf{v}^T + \mathbf{u}_0 \otimes \mathbf{v}_0$. P_1 computes $W_1 = \mathbf{u}_1 \otimes \mathbf{v}_1 - R_\mathbf{u} - R_\mathbf{v}^T$.

Outputs: P_0 outputs $(\mathbf{u}_0, \mathbf{v}_0, W_0)$, P_1 outputs $(\mathbf{u}_1, \mathbf{v}_1, W_1)$.

Fig. 2. RLWE-based tensor triple generation

sVOLE-Based Two-Party Tensor Triple Generation An OT-based protocol for generating Beaver multiplication triples was proposed in [19]. In [16], 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 $\mathscr{F}^{p,m,n}_{\text{RsVOLE}'}$, as shown in Figure 3.

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

Players: Participants S, R.
Functionality: Receive $(\mathsf{sender}, \mathsf{sid})$ from S and $(\mathsf{receiver}, \mathsf{sid})$ from R, sample $\mathbf{x} \in_r \mathbb{F}_p^m, V \in_r \mathbb{F}_p^{mn}, \mathbf{y} \in_r \mathbb{F}_p^n$, compute $U = \mathbf{x} \otimes \mathbf{y} + V \in \mathbb{F}_p^{mn}$, return $(\mathbf{x}, V)$ to S and $(\mathbf{y}, U)$ to R.

Fig. 3. Functionality of RsVOLE'

Given a protocol $\mathsf{RsVOLE}(m, n, \lambda)$ realizing $\mathscr{F}^{p,m,n}_{\text{RsVOLE}'}$, we can easily formulate many ways to generate a tensor multiplication triple. Figure 5 shows a straightforward method.

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

Players: Participants $P_1, ..., P_k$.
Functionality: Receive (sid, i) from P_i, sample $\mathbf{u} \in_r \mathbb{F}_p^m, \mathbf{v} \in_r \mathbb{F}_p^n$, compute $W = \mathbf{u} \otimes \mathbf{v}$, securely secret share $\mathbf{u}, \mathbf{v}, W$ into $[\mathbf{u}]_i, [\mathbf{v}]_i, [W]_i$ and return to P_i the share $([\mathbf{u}]_i, [\mathbf{v}]_i, [W]_i)$.

Fig. 4. Functionality of tensor triple generation

sVOLE-based 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 $(\mathbf{u}_1 = \mathbf{x}', \mathbf{v}_1 = \mathbf{y}, W_1 = U - V' + \mathbf{x}' \otimes \mathbf{y})$.

Fig. 5. sVOLE-based tensor triple generation

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 $p = 2$, as shown in Figure 6. The idea is similar to the one implemented in [7], and we simply adapt it in tensor form to suit our need.

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 details and refer to [7] for brevity.

4 Secure Multi-Dimensional Arithmetic Evaluations

In this section we will discuss various vector operations which may be performed securely using Beaver's method. When $m = n = 1$, tensor triples are simply usual Beaver triples. Therefore they can also be used for basic arithmetics. We mainly discuss the multi-dimensional computations below.

4.1 Linear Operations and Dot Product

As expected, additions and operations involving constants may all be computed locally without any interaction by the homomorphicity. More specifically, the

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

Inputs: None.
Primitive: A random OT extension protocol and a pre-determined pseudorandom generator PRG.
Interacting Phase:

1. The receiver R and the sender S invoke the ROT extension protocol (preferably with extension). ROT protocol outputs to S a family of pairs $\{\mathsf{seed}_{i,0}, \mathsf{seed}_{i,1}\}_{i=1,\ldots,n}$ and to R its choices $\{y_i, \mathsf{seed}_{i,y_i}\}_{i=1,\ldots,n}$. We denote $\mathbf{y} = (y_1, ..., y_n)$. Additionally S samples a string $\mathbf{x} \leftarrow_\$ \{0,1\}^m$;
2. S computes $\mathbf{s}_i = \mathsf{PRG}(\mathsf{seed}_{i,0}) \oplus \mathsf{PRG}(\mathsf{seed}_{i,1}) \oplus \mathbf{x}$ for $i = 1, ..., n$ and sends all $\mathbf{s}_i$ to R;
3. R computes $\mathbf{u}_i = \mathsf{PRG}(\mathsf{seed}_{i,y_i}) \oplus y_i \mathbf{s}_i$ and formulates the matrix $U = (\mathbf{u}_1^T, ..., \mathbf{u}_n^T)$. S formulates the matrix $V = (\mathsf{PRG}(\mathsf{seed}_{i,0}))_{i=1,\ldots,n}^T$.

Outputs: R outputs $(\mathbf{y}, U)$, S outputs $(\mathbf{x}, V)$.

Fig. 6. COT-based RsVOLE

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 $\mathbf{a} \in K^n$, 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.

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 $\mathbf{a} - \mathbf{u} = \mathbf{s}, \mathbf{b} - \mathbf{v} = \mathbf{t}$. 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. The correctness can easily be verified by a direct computation.

4.2 Outer Product and Matrix Product

Figure 7 shows the ideal functionality for secure multi-party outer product computation. 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 $\mathbf{a} - \mathbf{u} = \mathbf{s}, \mathbf{b} - \mathbf{v} = \mathbf{t}$. Then similarly we could compute

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

Functionality $\mathscr{F}_{\mathrm{Out}}$

Players: Participants $P_1, ..., P_k$.
Functionality: Receive $(\mathsf{sid}, i, [\mathbf{u}]_i, [\mathbf{v}]_i)$ from P_i, recover $\mathbf{u}, \mathbf{v}$ from shares $(i, [\mathbf{u}]_i, [\mathbf{v}]_i)$, abort if the recovery fails, securely secret share $\mathbf{u} \cdot \mathbf{v}$ into $[\mathbf{u} \otimes \mathbf{v}]_i$ and return to P_i the share $[\mathbf{u} \otimes \mathbf{v}]_i$.

Fig. 7. Functionality of secure 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 $[\mathbf{a} \otimes \mathbf{b}]_i$.

Fig. 8. Secure Multi-Party Outer Product

Therefore we could similarly make a protocol described below in Figure 8.

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.

Functionality $\mathscr{F}_{\mathrm{MatProd}}$

Players: Participants $P_1, ..., P_k$.
Functionality: Receive $(\mathsf{sid}, i, [A]_i, [B]_i)$ from P_i, recover A, B from shares $(i, [A]_i, [B]_i)$, abort if the recovery fails or A, B are not multiplicable, securely secret share $A \cdot B$ into $[AB]_i$ and return to P_i the share $[AB]_i$.

Fig. 9. Functionality of secure 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. Figure 9 shows the functionality of secure multi-party matrix multiplication. In practice, we consider $A \in M_{m\times k}(K), \quad B \in M_{k\times n}(K)$. 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 in Figure 10.

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 outer product $\mathsf{Out}(\mathbf{a}, \mathbf{b})$.
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 $[AB]_i \leftarrow \sum_{l=1}^{k} [\mathbf{a}_l \otimes \mathbf{b}_l]_i$.

Fig. 10. Secure Multi-Party Matrix Product

4.3 Security

On the security of all protocols proposed above, the proofs are somehow evident from the simple observation that the triple always hides perfectly the input vectors and in none of the protocols the third matrix share is published in any sense. This also illustrates the correct and secure usage of tensor triples, namely to randomize the input vectors to announce, while always keeping the third matrix share secret. The precise statements of the security are given below.

Theorem 5 ([39]). *The RLWE-based tensor triple generation protocol (Figure 2) realizes the functionality $\mathscr{F}_{TTGen}^{p,m,n}$ in the two-party setting and is semi-honest secure.*

Theorem 6 ([7]). *The COT-based RsVOLE protocol (Figure 6) realizes the functionality $\mathscr{F}_{RsVOLE'}^{p,m,n}$ in the two-party setting and is semi-honest secure.*

Theorem 7. *The sVOLE-based tensor triple generatio protocol* $\mathsf{TT.Gen}()$ *realizes the functionality $\mathscr{F}_{TTGen}^{p,m,n}$ in the two-party setting and is semi-honest secure in the $\mathscr{F}_{RsVOLE'}^{p,m,n}$-hybrid model.*

Theorem 8. *The secure outer product protocol* $\mathsf{Out}()$ *realizes the functionality $\mathscr{F}_{Out}$ and is semi-honest secure in the $\mathscr{F}_{TTGen}^{p,m,n}$-hybrid model.*

Theorem 9. *The secure matrix product protocol* $\mathsf{MatProd}()$ *realizes the functionality $\mathscr{F}_{MatProd}$ and is semi-honest secure in the $\mathscr{F}_{Out}$-hybrid model.*

Proof. The proofs of the three theorems 7,8,9 are given in details in Appendix A.

4.4 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 1024×1024 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 [39] 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 Sec. 5. 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.

5 Applications

5.1 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.

5.2 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.

FingerCode FingerCode [24,22] 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 $D_{ij} \leq d$;

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.

Eigenfaces Eigenfaces [44] 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 $D_{ij} \leq d$;

FaceNet FaceNet [41] 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 $D_{ij} \leq d$;

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, 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.

6 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 10 times and the medians of the results are presented in the following tables. Source codes have been provided at https://github.com/lzjluzijie/triple.

6.1 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 [38] 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.

We present the performance as well as the corresponding communication cost for each method in Table 1,2,3. 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).

$m \backslash n$	2^3		2^8		2^{10}		2^{14}	
	LAN	WAN	LAN	WAN	LAN	WAN	LAN	WAN
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		

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 [39], 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.

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).

$m\backslash n$	2^3		2^8		2^{10}		2^{14}	
	LAN	WAN	LAN	WAN	LAN	WAN	LAN	WAN
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 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).

$m\backslash n$	2^3		2^8		2^{10}		2^{14}	
	COT	SOT	COT	SOT	COT	SOT	COT	SOT
2^3	0.032	1.00	0.52	1.00	2.02	1.00	32.02	1.23
	0.76	32.86	16.26	32.86	64.26	32.86	1024.26	39.31
	12.10	525.82	260.10	525.82	1028.10	525.82	16388.08	628.95
2^8			16.26	28.79	64.26	28.79	1024.26	28.99
			520.01	921.21	2056.01	921.21	32776.01	927.65
			8320.08	14739.36	32896.08	14739.36	-	14842.48
2^{10}					257.01	114.76	4097.02	114.96
					8224.01	3672.21	131104.04	3678.65
					130969.60	58755.36	-	58858.48

6.2 Matrix Multiplication

Table 4 shows the performance and the communication cost of the online phase for our implementation of the tensor triple based secure multi-party matrix multiplication protocol.

As a comparison with previous works [14,11,31], we also provide the following table on the performance of secure multiparty square matrix multiplication protocols. Due to a lack of source codes or problems in code running, we were not able to individually launch experiments in these previous works under the same environment, but we believe the statistics in the table already implies the high efficiency of the tensor triple method. Note that our experiments are executed with a single thread, and matrix multiplication is known to be highly parallelizable. The computation can be fulfilled easily using parallel computation. Although not given explicitly in the performance tables, parallel computation is capable of significantly reducing the overall computational cost. Hence our performance significantly outperforms all previous works listed in the table. This is reasonable as in these previous works homomorphic encryption is heavily used,

Table 4. Performance and communication cost of tensor triple-based 32-bit $(m, k) \times (k, n)$ matrix multiplication time (in milliseconds and megabytes, resp.).

$k \backslash (m, n)$	$(2^3, 2^3)$			$(2^5, 2^5)$			$(2^8, 2^8)$			$(2^{10}, 2^{10})$		
	LAN	WAN	Com.	LAN	WAN	Com.	LAN	WAN	Com.	LAN	WAN	Com.
2^3	1	26	0.001	1	26	0.004	1	27	0.031	11	44	0.125
2^8	1	26	0.031	1	37	0.125	22	62	1	292	323	4
2^{10}	1	37	0.125	2	44	0.5	71	128	4	1062	1289	16

while in our work we have mainly applied VOLE which is somewhat lightweight comparatively.

Table 5. Comparison of our performance with previous works on 128-bit sqaure matrix multiplication of size $d \times d$ (Single thread by default, LAN environment, in seconds)

Size d	[11] (16 threads)	[11]	SPDZ [14]	[31] (16 threads)	Ours	Ours (Offline)
128	5.9	36.1	128	3.09	**0.007**	0.51
256	25.5	214.5	900	13.49	**0.049**	2.82
384	68.3	653.6	2808	33.6	**0.158**	9.22
512	138	1470	6300	67.39	**0.402**	18.94
1024	870	10380	44100	395.2	**4.324**	143.36

6.3 Batched Privacy-Preserving Biometric Identification

In this section we present performance of tensor triple based implementations of batched queries of FingerCode [24] and Eigenfaces [44] protocols with a comparison of the efficiency with the GSHADE [9] protocol, and FaceNet [41] with a comparison with the [34] 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 6 shows a comparison between the performances of our FingerCode implementation with COT-based triple generation and the one in [9]. Clearly,

Table 6. Performance of secure squared Euclidean distance computation in batched FingerCode protocol (128 queries)

	$n = 128$			$n = 1024$		
Protocol	[9]	Ours	Ours (Offine)	[9]	Ours	Ours (Offline)
Time (s)	154.37	**0.016**	1.90	176.64	**0.082**	15.54
Comm. (MB)	1688.71	**1.25**	2640.02	5379.24	**5.63**	20560

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

Table 7. Performance of Eigenfaces protocol without the secure comparison step (80 queries)

	$N = 320$		$N = 1000$	
Protocol	Ours	Ours (Offline)	Ours	Ours (Offline)
Time (s)	0.029	41.44	0.032	124.24
Communication (MB)	14.57	65207	14.69	202057

Table 7 shows the performance of our implementation for Eigenfaces protocol with COT-based triple generation. As a comparison to the performance in [9], 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 [9] 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 8. Performance (time in seconds) of secure squared Euclidean distance computation in batched FaceNet protocol for m queries against a database of n references

(m, n)	[34]	Ours (Online)	Ours (Offline, COT)	Ours (Offline, SOT)
$(2^4, 2^4)$	2.58	**0.00068**	0.02	-
$(2^4, 2^{10})$	165.95	**0.0055**	0.378	12.00
$(2^{10}, 2^{10})$	10559.68	**0.25**	19.456	1219.05

Table 8 shows a comparison between the performances of our FaceNet implementation and the one in [34]. We achieve a significant speedup by around 10000 times.

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 shown in the tables. 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, there is no valid way to split the protocol into online and offline process, as the dimensions of the matrices is already involved in its fundamental components, such as OT. Therefore, it should be considered fair for such comparisons in the tables we listed.

7 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

1. Abspoel, M., Cramer, R., Damgrard, I., Escudero, D., Rambaud, M., Xing, C., Yuan, C.: 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. Applebaum, B., Damgrard, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 223–254 (2017)
3. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: 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)
4. Blanton, M., Gasti, P.: Secure and efficient protocols for iris and fingerprint identification. In: Atluri, V., Diaz, C. (eds.) Computer Security – ESORICS 2011. pp. 190–209 (2011)
5. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1175–1191 (2017)
6. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector ole. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 896–912 (2018)
7. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: 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
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. p. 309–325 (2012)
9. Bringer, J., Chabanne, H., Favre, M., Patey, A., Schneider, T., Zohner, M.: Gshade: Faster privacy-preserving distance computation and biometric identification. In: Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security. p. 187–198 (2014)

10. Bringer, J., Chabanne, H., Patey, A.: Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends. In: IEEE Signal Processing Magazine. pp. 42–52 (2013)
11. Chen, H., Kim, M., Razenshteyn, I.P., Rotaru, D., Song, Y., Wagh, S.: Maliciously secure matrix multiplication with applications to private deep learning. In: Advances in Cryptology - ASIACRYPT 2020. pp. 31–59 (2020)
12. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes. In: Advances in Cryptology – CRYPTO 2021, pp. 502–534 (2021)
13. Damg**r**ard, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Public Key Cryptography. pp. 119–136 (2001)
14. Damg**r**ard, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Advances in Cryptology – CRYPTO 2012. pp. 643–662 (2012)
15. Damg**r**ard, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for online auctions. In: Proceedings of the 12th Australasian Conference on Information Security and Privacy. p. 416–430 (2007)
16. Demmler, D., Schneider, T., Zohner, M.: Aby - a framework for efficient mixed-protocol secure two-party computation. In: Network and Distributed System Security Symposium (2015)
17. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Privacy Enhancing Technologies. pp. 235–253 (2009)
18. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012)
19. Gilboa, N.: Two party rsa key generation. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 116–129 (1999)
20. Gomez-Barrero, M., Galbally, J., Morales, A., Fierrez, J.: Privacy-preserving comparison of variable-length data with application to biometric template protection. IEEE Access **5**, 8606–8619 (2017)
21. Hahn, C., Hur, J.: Efficient and privacy-preserving biometric identification in cloud. ICT Express **2**(3), 135–139 (2016)
22. Huang, Y., Malka, L., Evans, D., Katz, J.: Efficient privacy-preserving biometric identification. Network and Distributed System Security Symposium (2011)
23. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Advances in Cryptology - CRYPTO 2003. pp. 145–161 (2003)
24. Jain, A.K., Prabhakar, S., Hong, L., Pankanti, S.: Fingercode: a filterbank for fingerprint representation and matching. Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149) **2**, 187–193 (1999)
25. Kolesnikov, V., Kumaresan, R.: Improved ot extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 54–70 (2013)
26. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: 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)
27. Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: Super-fast and robust Privacy-Preserving machine learning. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2651–2668 (2021)

28. Ma, Z., Liu, Y., Liu, X., Ma, J., Ren, K.: Lightweight privacy-preserving ensemble classification for face recognition. IEEE Internet of Things Journal **6**(3), 5778–5790 (2019)
29. Mohassel, P., Rindal, P.: Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 35–52 (2018)
30. Mohassel, P., Rosulek, M., Trieu, N.: Practical privacy-preserving k-means clustering. Proceedings on Privacy Enhancing Technologies **2020**, 414–433 (2020)
31. Mono, J., Güneysu, T.: Implementing and optimizing matrix triples with homomorphic encryption. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security. pp. 29–40 (2023)
32. Muth, P., Katzenbeisser, S.: Assisted mpc. Cryptology ePrint Archive, Paper 2022/1453 (2022), https://eprint.iacr.org/2022/1453
33. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM Journal on Computing **35**(5), 1254–1281 (2006)
34. Naresh Boddeti, V.: Secure face matching using fully homomorphic encryption. In: 2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS). pp. 1–10 (2018)
35. Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacy-preserving ridge regression on hundreds of millions of records. In: IEEE Symposium on Security and Privacy. pp. 334–348 (2013)
36. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 223–238 (1999)
37. Rabin, M.O.: How to exchange secrets with oblivious transfer. TR-81 edition (1981)
38. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-convolute codes for pseudorandom correlation generators from lpn. In: Advances in Cryptology – CRYPTO 2023. pp. 602–632 (2023)
39. Rathee, D., Schneider, T., Shukla, K.K.: Improved multiplication triple generation over rings via rlwe-based ahe. In: Cryptology and Network Security. pp. 347–359 (2019)
40. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-ole: Improved constructions and implementation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 1055–1072 (2019)
41. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 815–823 (2015)
42. Shahandashti, S.F., Safavi-Naini, R., Ogunbona, P.: Private fingerprint matching. In: Information Security and Privacy. pp. 426–433 (2012)
43. Smart, N.P., Tanguy, T.: Taas: Commodity mpc via triples-as-a-service. Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop (2019)
44. Turk, M., Pentland, A.: Eigenfaces for Recognition. Journal of Cognitive Neuroscience **3**(1), 71–86 (1991)
45. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: 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)
46. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167 (1986)

APPENDIX

A Security

In this chapter we give the detailed proofs for the security of the protocols.

Theorem. *The sVOLE-based tensor triple generatio protocol* TT.Gen() *realizes the functionality* $\mathscr{F}_{TTGen}^{p,m,n}$ *in the two-party setting and is semi-honest secure in the* $\mathscr{F}_{RsVOLE'}^{p,m,n}$*-hybrid model.*

Proof. Due to symmetry, it suffices to consider a semi-honest adversary P_0. The simulator plays the role of $\mathscr{F}_{\mathrm{RsVOLE}'}^{p,m,n}$ and provides P_0 with $\mathbf{x} \in_r \mathbb{F}_p^m, \mathbf{y}' \in_r \mathbb{F}_p^n, V \in_r \mathbb{F}_p^{mn}, U' \in_r \mathbb{F}_p^{mn}$, and itself as P_1 with $\mathbf{x}' \in_r \mathbb{F}_p^n, \mathbf{y} \in_r \mathbb{F}_p^n$. Then it computes $U = V + \mathbf{x} \otimes \mathbf{y}$ and $V' = U' - \mathbf{x}' \otimes \mathbf{y}'$. Since in the original interaction $\mathbf{x}, \mathbf{y}, V, U'$ are uniformly distributed over the corresponding ambient spaces, this hybrid is indistinguishable from the original interaction. Thus the protocol is secure against semi-honest P_0. The proof goes exactly the same way for a semi-honest adversary P_1.

Theorem. *The secure outer product protocol* Out() *realizes the functionality* $\mathscr{F}_{Out}$ *and is semi-honest secure in the* $\mathscr{F}_{TTGen}^{p,m,n}$*-hybrid model.*

Proof. This is obvious as the vector shares are perfectly rerandomized by the tensor triples and thus uniformly distributed over the corresponding ambient spaces. We emphasize the matrix share $[W]$ is never published in any of the protocols throughout the paper as it shall not be, hence not affecting the protocol security.

Theorem. *The secure matrix product protocol* MatProd() *realizes the functionality* $\mathscr{F}_{MatProd}$ *and is semi-honest secure in the* $\mathscr{F}_{Out}$*-hybrid model.*

Proof. As pointed out in the earlier sections, $\mathscr{F}_{\mathrm{MatProd}}$ can be seen as a repetitive application of $\mathscr{F}_{\mathrm{Out}}$. Therefore MatProd() as an application of multiple Out() individually realizes $\mathscr{F}_{\mathrm{MatProd}}$.

B Additively Homomorphic Encryption (AHE)

In this section we give a detailed defintion of an AHE scheme.

Definition 10. *An AHE scheme is a tuple of algorithms AHE=(Gen, Enc, Dec, Add, ScMult, ScTensor) described as follows:*

- *Gen*$(1^\lambda) \to$ (*pk,sk*)*: Key Generation is a randomized algorithm that outputs a pair of keys (pk,sk), with public key pk and secret key sk.*
- *Enc*$(\mathsf{pk}, m) \to$ *ct: Encryption is a randomized algorithm that takes a message* $m \in PT_{n,\lambda}$ *and the public key pk as input, and outputs a ciphertext ct* $\in CT_{n,\lambda}$*, where* $PT_{n,\lambda}$ *denotes the plaintext space of AHE for security parameter* λ *and rank* n*, and* $CT_{n,\lambda}$ *the corresponding ciphertext space.*

- $\mathsf{Dec}(sk,ct) \to m$: *Decryption is a deterministic algorithm that takes the secret key* sk *and a ciphertext* 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* pk *as input, and outputs a ciphertext* $\mathsf{ct}_+ \in CT_{n,\lambda}$. *This binary operation with respect to* $\mathsf{ct}_1, \mathsf{ct}_2$ *will be denoted by* $+$.
- $\mathsf{ScMult}(pk,ct,s) \to \mathsf{ct}_\bullet$: *Scalar Multiplication takes a ciphertext* $ct \in CT_{n,\lambda}$, *a plaintext* $s \in PT_{n,\lambda}$ *and the public key* pk *as input, and outputs a ciphertext* $\mathsf{ct}_\bullet \in CT_{n,\lambda}$. *This binary operation with respect to* ct *and* s *will be denoted by* $\bullet$.
- $\mathsf{ScTensor}(pk,ct,\boldsymbol{s}) \to \mathbf{ct}_\otimes$: *Scalar Tensor takes a ciphertext* $ct \in CT_{n,\lambda}$, *a constant vector* $\boldsymbol{s}$ *of dimension* l *and the public key* pk *as input, and outputs an array of ciphertexts* $\mathbf{ct}_\otimes \in CT^l_{n,\lambda}$. *This binary operation with respect to* 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* $(pk,sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and any message* m, *with an overwhelming probability we have*
   $$\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$$
   - *For a generic pair of keys* $(pk,sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and any two ciphertexts* $\mathsf{ct}_1, \mathsf{ct}_2$, *with an overwhelming probability we have*
   $$\mathsf{Dec}(sk, \mathsf{Add}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)) = \mathsf{Dec}(sk, \mathsf{ct}_1) + \mathsf{Dec}(sk, \mathsf{ct}_2)$$
   - *For a generic pair of keys* $(pk,sk) \leftarrow \mathsf{Gen}(1^\lambda)$, *a ciphertext* ct *and a plaintext scalar* s, *with an overwhelming probability we have*
   $$\mathsf{Dec}(sk, \mathsf{ScMult}(\mathsf{pk}, \mathsf{ct}, s)) = s\mathsf{Dec}(sk, \mathsf{ct})$$
   - *For a generic pair of keys* $(pk,sk) \leftarrow \mathsf{Gen}(1^\lambda)$, *a ciphertext* ct *and a scalar vector* $\boldsymbol{s}$, *with an overwhelming probability we have*
   $$\mathsf{Dec}(sk, \mathsf{ScTensor}(\mathsf{pk}, \mathsf{ct}, \boldsymbol{s})) = \boldsymbol{s} \otimes \mathsf{Dec}(sk, \mathsf{ct}),$$
   where the decryption procedure is applied to each row of the ciphertext array.
2. *Security: The scheme is required to be IND-CPA secure.*

C 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 [43,32]. 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 may not need to 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. The detail of the third-party generation will be provided in a follow-up paper.