Parallel Multi-Party Computation from Linear Multi-Secret Sharing Schemes *

Zhifang Zhang[1], Mulan Liu[1]**, and Liangliang Xiao[2]

[1] Academy of Mathematics and Systems Science, Key Laboratory of Mathematics Mechanization, Chinese Academy of Sciences, Beijing, 100080, China
{zfz, mlliu}@amss.ac.cn
[2] Institute of Software, Chinese Academy of Sciences, Beijing, 100080, China
xllemail2004@yahoo.com.cn

Abstract. As an extension of multi-party computation (MPC), we propose the concept of secure parallel multi-party computation which is to securely compute multi-functions against an adversary with multi-structures. Precisely, there are m functions $f_1, ..., f_m$ and m adversary structures $\mathcal{A}_1, ..., \mathcal{A}_m$, where f_i is required to be securely computed against an $\mathcal{A}_i$-adversary. We give a general construction to build a parallel multi-party computation protocol from any linear multi-secret sharing scheme (LMSSS), provided that the access structures of the LMSSS allow MPC at all. When computing complicated functions, our protocol has more advantage in communication complexity than the "direct sum" method which actually executes a MPC protocol for each function. The paper also provides an efficient and generic construction to obtain from any LMSSS a multiplicative LMSSS for the same multi-access structure.

1 Introduction

The secure multi-party computation (MPC) protocol is used for n players to jointly compute an agreed function of their private inputs in a secure way, where security means guaranteeing the correctness of the output and the privacy of the players' inputs, even when some players cheat. It is fundamental in cryptography and distributed computation, because a solution of MPC problem implies in principle a solution to any cryptographic protocol problem, such as the voting problem, blind signature, and so on. After it was proposed by Yao [11] for two-party case and Goldreich, Micali, Wigderson [6] for multi-party case, it has become an active and developing field of information security.

In the MPC problem, it is common to model cheating by considering an adversary who may corrupt some subset of the players. The collection of all subsets that an adversary may corrupt is called the *adversary structure*, denoted by $\mathcal{A}$, and this adversary is called an $\mathcal{A}$-adversary. So the MPC problem is to securely

* Supported by the National Natural Science Foundation of China (No. 90304012, 90204016), 973 project (No. 2004CB318000).
** Corresponding author Mulan Liu. Email: mlliu@amss.ac.cn

compute a function with respect to an adversary structure. But in practice it is sometimes needed to simultaneously compute several different functions with respect to different adversary structures, respectively. For example, in the voting problem $n = 2t+1$ ($t > 1$) voters are to select a chairman and several fellows for a committee at the same time from m candidates. Because the position of the chairman is more important than that of fellows, the voting for the chairman is required to be secure against a (t, n) threshold adversary, while the voting for the fellows is required to be secure against a $(2, n)$ threshold adversary. Hence it makes us to propose *parallel multi-party computation* or extend MPC to parallel MPC. Precisely, in the problem of parallel multi-party computation, there are m functions $f_1, ..., f_m$ and m adversary structures $\mathcal{A}_1, ..., \mathcal{A}_m$, where f_i is required to be securely computed against an $\mathcal{A}_i$-adversary.

Obviously, secure parallel multi-party computation can be realized by designing for each function a MPC protocol with respect to the corresponding adversary structure, and then running all the protocols in a composite way. We call this the "direct sum" method. In this paper, we propose another way to realize parallel multi-party computation. It is well known that secret sharing schemes are elementary tool for studying MPC. Cramer, Damgard, Maurer [3] gave a generic and efficient construction to build a MPC protocol from any linear secret sharing scheme (LSSS). As an extension of secret sharing schemes, Blundo, De Santis, Di Crescenzo [2] proposed the general concept of *multi-secret sharing schemes* which is to share multi-secrets with respect to multi-access structures, and Ding, Laihonen, Renvall. [4] studied linear multi-secret sharing schemes. Based on Xiao and Liu's work [10] about linear multi-secret sharing schemes (LMSSS) and the construction in [3], we give a generic and efficient construction to build a parallel multi-party computation protocol from any LMSSS, provided that the access structures of the LMSSS allow MPC at all [7]. We only deal with *adaptive, passive* adversaries in the *information theoretic model*. When computing complicated functions, our protocol has more advantage in communication complexity than the "direct sum" method.

The paper is organized as follows: in Section 2 we review some basic concepts, such as LSSS, monotone span programs (MSP) and LMSSS. In Section 3 we give a clear description for the problem of secure parallel multi-party computation, and then obtain a generic protocol for it from any LMSSS. Furthermore we compare our protocol with the "direct sum" method in communication complexity. In the last section, a specific example is displayed in detail to show how our protocol works as well as its advantage.

2 Preliminaries

Since secret sharing schemes are our primary tool, first we review some basic concepts and results about them, such as linear secret sharing schemes, multi-secret sharing schemes, monotone span programs, and so on. Suppose that $P = \{P_1, ..., P_n\}$ is the set of participants and $\mathcal{K}$ is a finite field throughout this paper.

2.1 LSSS vs MSP

It is well-known that an access structure, denoted by AS, is a collection of subsets of P satisfying the monotone ascending property: for any $A' \in AS$ and $A \in 2^P$ with $A' \subset A$, it holds that $A \in AS$; and an adversary structure, denoted by $\mathcal{A}$, is a collection of subsets of P satisfying the monotone descending property: for any $A' \in \mathcal{A}$ and $A \in 2^P$ with $A \subset A'$, it holds that $A \in \mathcal{A}$. In this paper, we consider the complete situation, i.e. $\mathcal{A} = 2^P - AS$. Because of the monotone property, for any access structure AS it is enough to consider the *minimum access structure* AS_m defined as $AS_m = \{A \in AS \mid \forall B \subsetneqq A \Rightarrow B \notin AS\}$.

Suppose that S is the secret-domain, R is the set of random inputs, and S_i is the share-domain of P_i where $1 \leq i \leq n$. A secret sharing scheme with respect to an access structure AS is composed of the distribution function $\Pi : S \times R \to S_1 \times \cdots \times S_n$ and the reconstruction function: for any $A \in AS$, $Re = \{Re_A : (S_1 \times \cdots \times S_n)|_A \to S \mid A \in AS\}$, such that the following two requirements are satisfied.

(i) Correctness requirement: for any $A \in AS, s \in S$ and $r \in R$, it holds that $Re_A(\Pi(s,r)|_A) = s$, where suppose $A = \{P_{i_1}, ..., P_{i_{|A|}}\}$ and $\Pi(s,r) = (s_1, ..., s_n)$, then $\Pi(s,r)|_A = (s_{i_1}, ..., s_{i_{|A|}})$.

(ii) Security requirement: for any $B \notin AS$, *i.e.*, $B \in \mathcal{A} = 2^P \setminus AS$, it holds that $0 < H(S|\Pi(S,R)|_B) \leq H(S)$, where $H(\cdot)$ is the entropy function.

In the security requirement, if $H(S|\Pi(S,R)|_B) = H(S)$, we call it a perfect secret sharing scheme which we are interested in. Furthermore, a perfect secret sharing scheme is *linear* (LSSS for short), if S, R, S_i are all linear spaces over $\mathcal{K}$ and the reconstruction function is linear [1].

Karchmer and Wigderson [8] introduced monotone span programs (MSP) as linear models computing monotone Boolean functions. Usually we denote a MSP by $\mathcal{M}(\mathcal{K}, M, \psi)$, where M is a $d \times l$ matrix over $\mathcal{K}$ and $\psi : \{1, ..., d\} \to \{P_1, ..., P_n\}$ is a surjective labelling map which actually distributes to each participant some rows of M. We call d the *size* of the MSP. For any subset $A \subseteq P$, there is a corresponding characteristic vector $\overrightarrow{\delta_A} = (\delta_1, ..., \delta_n) \in \{0,1\}^n$ where for $1 \leq i \leq n$, $\delta_i = 1$ if and only if $P_i \in A$. Consider a monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ which satisfies that for any $A \subseteq P$ and $B \subseteq A$, $f(\overrightarrow{\delta_B}) = 1$ implies $f(\overrightarrow{\delta_A}) = 1$. We say that a MSP $\mathcal{M}(\mathcal{K}, M, \psi)$ computes the monotone Boolean function f with respect to a target vector $\overrightarrow{v} \in \mathcal{K}^l \setminus \{(0, ..., 0)\}$, if it holds that $\overrightarrow{v} \in span\{M_A\}$ if and only if $f(\overrightarrow{\delta_A}) = 1$, where M_A consists of the rows i of M with $\psi(i) \in A$ and $\overrightarrow{v} \in span\{M_A\}$ means that there exists a vector $\overrightarrow{w}$ such that $\overrightarrow{v} = \overrightarrow{w} M_A$. Beimel [1] proved that devising a LSSS with respect to an access structure AS is equivalent to constructing a MSP computing the monotone Boolean function f_{AS} which satisfies $f_{AS}(\overrightarrow{\delta_A}) = 1$ if and only if $A \in AS$.

2.2 LMSSS vs MSP

Multi-Secret sharing schemes [2] are to share multi-secrets with respect to multi-access structures. Precisely, let $AS_1, ..., AS_m$ be m access structures over P,

$S^1 \times \cdots \times S^m$ be the secret-domain,$S_1, ..., S_n$ be the share-domain and R be the set of random inputs. Without loss of generality, we assume that $S^1 = \cdots = S^m = \mathcal{K}$. A linear multi-secret sharing scheme (LMSSS for short) realizing the multi-access structure $AS_1, \cdots, AS_m$ is composed of the distribution function

$$\Pi : \mathcal{K}^m \times R \longrightarrow S_1 \times \cdots \times S_n$$

$$\Pi(s^1, \cdots, s^m, r) = (\Pi_1(s^1, \cdots, s^m, r), \cdots, \Pi_n(s^1, \cdots, s^m, r)), \qquad (1)$$

and the reconstruction function $Re = \{Re_A^i : (S_1 \times \cdots \times S_n)_A \to \mathcal{K} | 1 \leq i \leq m,\ A \in AS_i\}$, such that the following three conditions hold:

(i) $S_1, \cdots, S_n$ and R are finitely dimensional linear spaces over $\mathcal{K}$, *i.e.*, there exist positive integers d_k, $1 \leq k \leq n$, and l such that $S_k = \mathcal{K}^{d_k}$ and $R = \mathcal{K}^l$. Precisely, in the equality (1), we have that $\Pi_k(s^1, \cdots, s^m, r) \in \mathcal{K}^{d_k}$ for $1 \leq k \leq n$. Furthermore, denote

$$\Pi_k(s^1, \cdots, s^m, r) = (\Pi_{k1}(s^1, \cdots, s^m, r), \cdots, \Pi_{kd_k}(s^1, \cdots, s^m, r))$$

where $\Pi_{kj}(s^1, \cdots, s^m, r) \in \mathcal{K}$ and $1 \leq j \leq d_k$. Usually $d = \sum_{i=1}^n d_i$ is called the size of the linear multi-secret sharing scheme.

(ii) The reconstruction function is linear. That is, for any set $A \in AS_i$, $1 \leq i \leq m$, there exists a set of constants $\{\alpha_{kj}^i \in \mathcal{K} | 1 \leq k \leq n, P_k \in A, 1 \leq j \leq d_k\}$ such that for any $s^1, ..., s^m \in \mathcal{K}$ and $r \in R$, $s^i = Re_A^i(\Pi(s^1, ..., s^m, r)|_A) = \sum_{p_k \in A} \sum_{j=1}^{d_k} \alpha_{kj}^i \Pi_{kj}(s^1, \cdots s^m, r)$.

(iii) Security requirement: For any set $B \subset \{P_1, \cdots, P_n\}, T \subset \{S^1, \cdots, S^m\} \backslash \{S^i | B \in AS_i, 1 \leq i \leq m\}$, it holds that $H(T|B) = H(T)$, where $H(\cdot)$ is the entropy function.

Similar to the equivalence relation of LSSS and MSP, Xiao and Liu [10] studied a corresponding relation between LMSSS and MSP computing multi-Boolean functions. Let $\mathcal{M}(\mathcal{K}, M, \psi)$ be a MSP with the $d \times l$ matrix M and $f_1, ..., f_m : \{0,1\}^n \to \{0,1\}$ be m monotone Boolean functions. Suppose $\overrightarrow{v_1}, ..., \overrightarrow{v_m}$ are m linear independent l-dimension vectors over $\mathcal{K}$, then it follows that $m \leq l$. In practice, we always have $m < l$ in order to use randombits. Then $\mathcal{M}$ can compute the Boolean functions $f_1, ..., f_m$ with respect to $\overrightarrow{v_1}, ..., \overrightarrow{v_m}$ if for any $1 \leq k \leq m$ and $1 \leq i_1 < \cdots < i_k \leq m$, the following two conditions hold:

(i) For any $A \subseteq P$, $f_{i_1}(\overrightarrow{\delta_A}) = \cdots = f_{i_k}(\overrightarrow{\delta_A}) = 1$ implies that $\overrightarrow{v_{i_j}} \in span\{M_A\}$ for $1 \leq j \leq k$.

(ii) For any $A \subseteq P$, $f_{i_1}(\overrightarrow{\delta_A}) = \cdots = f_{i_k}(\overrightarrow{\delta_A}) = 0$ implies that $Rank \begin{pmatrix} M_A \\ \overrightarrow{v_{i_1}} \\ \vdots \\ \overrightarrow{v_{i_k}} \end{pmatrix} =$ $Rank\ M_A + k$.

After a proper linear transform, any MSP computing the multi-Boolean function $f_{AS_1}, \cdots, f_{AS_m}$ with respect to $\overrightarrow{v_1}, ..., \overrightarrow{v_m}$ can be converted into a MSP computing the same multi-Boolean function with respect to $\overrightarrow{e_1}, \cdots, \overrightarrow{e_m}$, where $\overrightarrow{e_i} = (0, ..., 0, \overset{i}{1}, 0, ..., 0) \in \mathcal{K}^l$ for $1 \leq i \leq m$. So without loss of generality we always assume the target vectors are $\overrightarrow{e_1}, \cdots, \overrightarrow{e_m}$.

Theorem 1. *[10] Let $AS_1, \cdots, AS_m$ be m access structures over P and $f_{AS_1}, \cdots, f_{AS_m}$ be the corresponding characteristic functions. Then there exists a linear multi-secret sharing scheme realizing $AS_1, \cdots, AS_m$ over a finite field $\mathcal{K}$ with size d if and only if there exists a monotone span program computing monotone Boolean functions $f_{AS_1}, \cdots, f_{AS_m}$ with size d.*

Actually, let $\mathcal{M}(\mathcal{K}, M, \psi)$ be a MSP computing monotone Boolean functions $f_{AS_1}, \cdots, f_{AS_m}$ with respect to $\overrightarrow{e_1}, \cdots, \overrightarrow{e_m}$, where M is a $d \times l$ matrix. Then the corresponding LMSSS realizing $AS_1, \cdots, AS_m$ over $\mathcal{K}$ is as follows: For any multi-secret $(s^1, ..., s^m) \in \mathcal{K}^m$ and random input $\overrightarrow{\rho} \in \mathcal{K}^{l-m}$, the distribution function is defined by

$$\Pi(s^1, \cdots, s^m, \overrightarrow{\rho}) = ((s^1, \cdots, s^m, \overrightarrow{\rho})(M_{P_1})^\tau, \cdots, (s^1, \cdots, s^m, \overrightarrow{\rho})(M_{P_n})^\tau),$$

where "τ" denotes the transpose and M_{P_k} denotes M restricted to those rows i with $\psi(i) = P_k$, $1 \leq i \leq d, 1 \leq k \leq n$. As to reconstruction, since $\overrightarrow{e_i} \in span\{M_A\}$ for any $A \in AS_i$, *i.e.*, there exists a vector $\overrightarrow{v}$ such that $\overrightarrow{e_i} = \overrightarrow{v} M_A$, then

$$s^i = (s^1, \cdots, s^m, \overrightarrow{\rho})\overrightarrow{e_i}^\tau = (s^1, \cdots, s^m, \overrightarrow{\rho})(\overrightarrow{v} M_A)^\tau = (s^1, \cdots, s^m, \overrightarrow{\rho})(M_A)^\tau \overrightarrow{v}^\tau,$$

where $(s^1, \cdots, s^m, \overrightarrow{\rho})(M_A)^\tau$ are the shares held by players in A and $\overrightarrow{v}$ can be computed by every participant.

3 Parallel Multi-Party Computation

3.1 Concepts and Notations

The problem of secure MPC for one function has been studied by many people and it can be stated as follows: n players $P_1, ..., P_n$ are to securely compute an agreed function $f(x_1, ..., x_n) = (y_1, ..., y_n)$ against an $\mathcal{A}$-adversary, where P_i holds private input x_i and is to get the output y_i. The security means that the correctness of the outputs and the privacy of players' inputs are always guaranteed no matter which set in $\mathcal{A}$ is corrupted by the adversary. In fact the function f can be represented as $f = (f_1, ..., f_n)$ where $f_i(x_1, ..., x_n) = y_i$ for $1 \leq i \leq n$. As the general way of treating the MPC problem, we assume that the functions involved thereafter are all of the form of f_i. So the MPC problem can be seemed as securely computing n functions with respect to the same adversary structure. As a natural extension, it is reasonable to consider securely computing multi-functions with respect to multi-adversary structures. Thus we propose the concept of *secure parallel multi-party computation*.

Precisely, there are m functions $f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n)$ and m corresponding adversary structures $\mathcal{A}_1, ..., \mathcal{A}_m$. For $1 \leq i \leq n$, player P_i has private input $(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(m)})$, where $x_i^{(j)}$ is P_i's input to the function $f_j(x_1, ..., x_n)$. So the final value of f_j is $f_j(x_1^{(j)}, x_2^{(j)}, ..., x_n^{(j)})$. An $(\mathcal{A}_1, ..., \mathcal{A}_m)$-adversary can corrupt any set in $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_m$. The n players are to securely compute the

multi-function $f_1, ..., f_m$ against an $(\mathcal{A}_1, ..., \mathcal{A}_m)$-adversary, that is, for any corrupted set $B \in \mathcal{A}_{i_1} \cap \cdots \cap \mathcal{A}_{i_k}$, where $1 \leq i_1 < \cdots < i_k \leq m$ and $k \leq m$, functions $f_{i_1}, ..., f_{i_k}$ are securely computed, which includes the following two aspects:

(i) Correctness: For $1 \leq i \leq n$, P_i finally gets the correct outputs of the functions $f_{i_1}, ..., f_{i_k}$.

(ii) Privacy: The adversary gets no information about other players' (players out of B) inputs for functions $f_{i_1}, ..., f_{i_k}$, except what can be implied from the inputs and outputs held by players in B.

The problem of secure parallel multi-party computation for the multi-function $f_1, ..., f_m$ against an $(\mathcal{A}_1, ..., \mathcal{A}_m)$-adversary is essentially a direct composition of problems of secure MPC for f_j against an $\mathcal{A}_j$-adversary where $1 \leq j \leq m$. So it can be resolved by designing for each function and the corresponding adversary structure a secure MPC protocol and running them in a composite way. We call this a "direct sum" method. One of the results in [7] tells us that in the information theoretic model, every function can be securely computed against an adaptive, passive $\mathcal{A}$-adversary if and only if $\mathcal{A}$ is Q2, where Q2 is the condition that no two of the sets in the structure cover the full player set. Thus we evidently have the following proposition.

Proposition 1. *In the information theoretic model, there exists a parallel multi-party computation protocol computing m functions securely against an adaptive, passive $(\mathcal{A}_1, ..., \mathcal{A}_m)$-adversary if and only if $\mathcal{A}_1, ..., \mathcal{A}_m$ are all Q2.*

Cramer et al. [3] build a secure MPC protocol for one function based on the *multiplicative* MSP computing one Boolean function. Here we extend it to the *multiplicative* MSP computing multi-Boolean functions. Precisely, let $\mathcal{M}(\mathcal{K}, M, \psi)$ be a MSP described in Section 2. Given two vectors $\overrightarrow{x} = (x_1, ..., x_d)$, $\overrightarrow{y} = (y_1, ..., y_d) \in \mathcal{K}^d$, we let $\overrightarrow{x} \diamond \overrightarrow{y}$ be the vector containing all entries of the form $x_i \cdot y_j$ with $\psi(i) = \psi(j)$, and $< \overrightarrow{x}, \overrightarrow{y} >$ denote the inner product. For example, let

$$\overrightarrow{x} = (x_{11}, ..., x_{1d_1}, ..., x_{n1}, ..., x_{nd_n}), \quad \overrightarrow{y} = (y_{11}, ..., y_{1d_1}, ..., y_{n1}, ..., y_{nd_n}),$$

where $\sum_{i=1}^{n} d_i = d$ and $x_{i1}, ..., x_{id_i}$, as well as $y_{i1}, ..., y_{id_i}$ are the entries distributed to P_i according to ψ. Then $\overrightarrow{x} \diamond \overrightarrow{y}$ is the vector composed of the $\sum_{i=1}^{n} d_i^2$ entries $x_{ij}y_{ik}$, where $1 \leq j, k \leq d_i, 1 \leq i \leq n$, and $< \overrightarrow{x}, \overrightarrow{y} >= \sum_{i=1}^{n} \sum_{j=1}^{d_i} x_{ij}y_{ij}$. Using these notations, we give the following definition.

Definition 1. *A monotone span program $\mathcal{M}(\mathcal{K}, M, \psi)$ computing Boolean functions $f_1, ..., f_m$ with respect to $\overrightarrow{e_1}, \cdots, \overrightarrow{e_m}$ is called multiplicative, if for $1 \leq i \leq m$, there exists a $\sum_{i=1}^{n} d_i^2$-dimensional recombination vector $\overrightarrow{r_i}$, such that for any two multi-secrets $(s^1, ..., s^m), (s'^1, ..., s'^m) \in \mathcal{K}^m$ and any $\overrightarrow{\rho}, \overrightarrow{\rho}' \in \mathcal{K}^{l-m}$, it holds that*

$$s^i s'^i = < \overrightarrow{r_i}, (s^1, ..., s^m, \overrightarrow{\rho})M^\tau \diamond (s'^1, ..., s'^m, \overrightarrow{\rho}')M^\tau > .$$

In fact, when $m = 1$ the definition is the same as that of [3]. In the appendix we give an efficient and generic construction to build from any MSP a multiplicative MSP computing the same multi-Boolean function. Hence in the following we assume that the based MSP in Section 3.2 is already multiplicative.

3.2 Construction from any LMSSS

In this section, assuming the adversary is passive and adaptive, we give a generic and efficient construction to obtain from any LMSSS a paralel multi-party computation protocol in the information theoretic model, provided that the access structures of the LMSSS allow MPC at all. Since LMSSS and MSP are equivalent, it turns out to be convenient to describe our protocol in terms of MSP's. We only describe the protocol in the case $m = 2$ and it is a natural extension for $m > 2$.

Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are two adversary structures over P and they are both Q2. For $1 \leq i \leq n$, player P_i has private input $(x_i^{(1)}, x_i^{(2)})$ and they are to jointly compute functions $f_1(x_1, ..., x_n)$ and $f_2(x_1, ...x_n)$. Let $AS_1 = 2^P \setminus \mathcal{A}_1,\ AS_2 = 2^P \setminus \mathcal{A}_2$, and $\mathcal{M}(\mathcal{K}, M, \psi)$ be a multiplicative MSP computing Boolean functions f_{AS_1} and f_{AS_2} with respect to target vectors $\overrightarrow{e_1}, \overrightarrow{e_2}$, where M is a $d \times l$ matrix over $\mathcal{K}$. How to construct such a MSP is out of concern in this paper. Next we describe our protocol in three phases: input sharing, computing and outputting.

Input Sharing. First each player shares his private input by using the MSP $\mathcal{M}(\mathcal{K}, M, \psi)$, *i.e.*, for $1 \leq i \leq n$, player P_i secretly and randomly selects $\overrightarrow{\rho_i}$ in the set of random inputs $R = \mathcal{K}^{l-2}$ and sends $(x_i^{(1)}, x_i^{(2)}, \overrightarrow{\rho_i})(M_{P_j})^\tau$ to player P_j, where $1 \leq j \leq n$ and $j \neq i$.

Computing. Since any function that is feasible to compute at all can be specified as a polynomial size arithmetic circuit over a finite field $\mathcal{K}$ with addition gates and multiplication gates, it is enough for us to discuss how to do additions and multiplications over $\mathcal{K}$. Different from computing a single function, in parallel multi-party computation, we compute the functions simultaneously other than one after another.

Precisely, suppose f_1 contains p multiplications and f_2 contains q multiplications, where $p \leq q$ and the multiplication considered here is operation between two elements. Then in each of the first p steps, we compute two multiplications coming from the two functions, respectively. In each the following $q - p$ steps, we continue to compute a multiplications of f_2 and do nothing for f_1. So after q steps we complete all the multiplications of both functions and get the intermediate results needed. Finally we compute all additions of both functions in one step. By doing so, we need less communication and random bits than the "direct sum" method. Furthermore, in order to guarantee security, all inputs and outputs of each step are multi-secret shared during computing and we call this condition the "invariant".

Example 1. Let $P = \{P_1, P_2, P_3\}$, and $f_1 = x_2^2 x_3,\ f_2 = x_1 x_2 + x_3$. For $1 \leq i \leq 3$, P_i has private input $(x_i^{(1)}, x_i^{(2)})$ which is multi-secret shared in the Input Sharing phase. Since f_1 contains two multiplications and f_2 contains one multiplication, the computing phase consists of three steps. The following table shows the computing process. Note that in the table, $x_i^{(j)}$ denotes an input value for the function f_j held by P_i, $z_i^{(j)}$ denotes an intermediate value held by an imaginary player

I_i, x_i and z_i are variables and z_{ij} is the function to be computed at each step, where $1 \le i \le 3$ and $1 \le j \le 2$.

	input	*to compute*	*output*
Step 1	$(x_1^{(1)}, x_1^{(2)})$ $(x_2^{(1)}, x_2^{(2)})$ $(x_3^{(1)}, x_3^{(2)})$	$(z_{11} = x_2x_3, z_{12} = x_1x_2)$	$(z_1^{(1)} = x_2^{(1)}x_3^{(1)}, z_1^{(2)} = x_1^{(2)}x_2^{(2)})$
Step 2	$(x_2^{(1)}, x_2^{(2)})$ $(z_1^{(1)}, z_1^{(2)})$	$(z_{21} = x_2z_1, z_{22} = z_1)$	$(z_2^{(1)} = x_2^{(1)}z_1^{(1)}, z_2^{(2)} = z_1^{(2)})$
Step 3	$(x_3^{(1)}, x_3^{(2)})$ $(z_2^{(1)}, z_2^{(2)})$	$(z_{31} = z_2, z_{32} = z_2 + x_3)$	$(z_3^{(1)} = z_2^{(1)}, z_3^{(2)} = z_2^{(2)} + x_3^{(2)})$

In Step 1 we do two multiplications x_2x_3 and x_1x_2 for f_1 and f_2, respectively; in Step 2 we do a multiplication x_2z_1 for f_1 and do nothing for f_2; in Step 3, we do an addition $z_2 + x_3$ for f_2 and do nothing for f_1. It is evident that $z_3^{(1)} = x_2^{(1)}x_2^{(1)}x_3^{(1)}$ and $z_3^{(2)} = x_1^{(2)}x_2^{(2)} + x_3^{(2)}$. The invariant here means that for $1 \le i \le 3$, $(x_i^{(1)}, x_i^{(2)})$, $(z_i^{(1)}, z_i^{(2)})$ all keep multi-secret shared by $\mathcal{M}(\mathcal{K}, M, \psi)$ during computing.

Next we discuss how to do multiplications or additions at each step. According to the type of operations we execute respectively for the two functions at each step (*e.g.* Step 1 of Example 1), there are four cases to be considered as follows, where " $\setminus$ " means that no operation is actually done and the output is one of the inputs. Without loss of generality, in the following we assume that $P = \{P_1, P_2, P_3, P_4\}$.

Case 1: $(+, +)$. First suppose that we are to compute $g_1 = x_1 + x_2$ and $g_2 = x_3 + x_4$. The inputs $(x_i^{(1)}, x_i^{(2)})$ are multi-secret shared such that each player P_j holds $(x_i^{(1)}, x_i^{(2)}, \overrightarrow{\rho_i})(M_{P_j})^\tau = (s_{i1}^{(j)}, ..., s_{id_j}^{(j)}) \in \mathcal{K}^{d_j}$ distributed by P_i where $1 \le i \le 4$. The output is to be multi-secret shared $(x_1^{(1)} + x_2^{(1)}, x_3^{(2)} + x_4^{(2)})$. Then P_j locally computes:

$$\begin{aligned} & (x_1^{(1)}, x_1^{(2)}, \overrightarrow{\rho_1})(M_{P_j})^\tau + (x_2^{(1)}, x_2^{(2)}, \overrightarrow{\rho_2})(M_{P_j})^\tau \\ &= (x_1^{(1)} + x_2^{(1)}, x_1^{(2)} + x_2^{(2)}, \overrightarrow{\rho_1} + \overrightarrow{\rho_2})(M_{P_j})^\tau \\ &= (s_{11}^{(j)} + s_{21}^{(j)}, ..., s_{1d_j}^{(j)} + s_{2d_j}^{(j)}) \,, \end{aligned} \tag{2}$$

$$\begin{aligned} & (x_3^{(1)}, x_3^{(2)}, \overrightarrow{\rho_3})(M_{P_j})^\tau + (x_4^{(1)}, x_4^{(2)}, \overrightarrow{\rho_4})(M_{P_j})^\tau \\ &= (x_3^{(1)} + x_4^{(1)}, x_3^{(2)} + x_4^{(2)}, \overrightarrow{\rho_3} + \overrightarrow{\rho_4})(M_{P_j})^\tau \\ &= (s_{31}^{(j)} + s_{41}^{(j)}, ..., s_{3d_j}^{(j)} + s_{4d_j}^{(j)}) \,. \end{aligned} \tag{3}$$

Actually, through (2) P_j gets shares for $(x_1^{(1)} + x_2^{(1)}, x_1^{(2)} + x_2^{(2)})$ and through (3) P_j gets shares for $(x_3^{(1)} + x_4^{(1)}, x_3^{(2)} + x_4^{(2)})$. In order to guarantee security, we

need to multi-secret share $(x_1^{(1)}+x_2^{(1)}, x_3^{(2)}+x_4^{(2)})$, each player must reshare his present shares. Precisely, by the reconstruction algorithm of the LMSSS, there exist $\overrightarrow{a}$, $\overrightarrow{b} \in \mathcal{K}^{\sum_{i=1}^{n} d_i}$, such that

$$x_1^{(1)}+x_2^{(1)} = \sum_{j=1}^{n}\sum_{k=1}^{d_j} a_{jk}(s_{1k}^{(j)}+s_{2k}^{(j)}), \quad x_3^{(2)}+x_4^{(2)} = \sum_{j=1}^{n}\sum_{k=1}^{d_j} b_{jk}(s_{3k}^{(j)}+s_{4k}^{(j)}) \ . \ (4)$$

So each player P_j reshares $(\sum_{k=1}^{d_j} a_{jk}(s_{1k}^{(j)}+s_{2k}^{(j)}), \sum_{k=1}^{d_j} b_{jk}(s_{3k}^{(j)}+s_{4k}^{(j)}))$ through $(\sum_{k=1}^{d_j} a_{jk}(s_{1k}^{(j)}+s_{2k}^{(j)}), \sum_{k=1}^{d_j} b_{jk}(s_{3k}^{(j)}+s_{4k}^{(j)}), \overrightarrow{\rho_j}')M^{\tau}$ and sends each of other players a share. Finally P_j adds up all his shares obtained from the resharing, *i.e.*,

$$\begin{aligned}
&\sum_{i=1}^{n}(\sum_{k=1}^{d_i} a_{ik}(s_{1k}^{(i)}+s_{2k}^{(i)}), \sum_{k=1}^{d_i} b_{ik}(s_{3k}^{(i)}+s_{4k}^{(i)}), \overrightarrow{\rho_i}')(M_{P_j})^{\tau} \\
&= (\sum_{i=1}^{n}\sum_{k=1}^{d_i} a_{ik}(s_{1k}^{(i)}+s_{2k}^{(i)}), \sum_{i=1}^{n}\sum_{k=1}^{d_i} b_{ik}(s_{3k}^{(i)}+s_{4k}^{(i)}), \sum_{i=1}^{n}\overrightarrow{\rho_i}')(M_{P_j})^{\tau} \\
&= (x_1^{(1)}+x_2^{(1)}, x_3^{(2)}+x_4^{(2)}, \sum_{i=1}^{n}\overrightarrow{\rho_i}')(M_{P_j})^{\tau} \ ,
\end{aligned}$$

which is actually P_j's share for $(x_1^{(1)}+x_2^{(1)}, x_3^{(2)}+x_4^{(2)})$.

Note that if we are to compute $(x_1^{(1)}+x_2^{(1)}, x_1^{(2)}+x_2^{(2)})$ at this step, the equality (2) is enough and we do not need resharing any more. Although we only discuss adding up two items here, we can add up more items once in the same way. Furthermore, it is trivial to deal with multiplications with constants in $\mathcal{K}$, since the constant is public.

Case 2: $(\times, \times)$. Suppose we are to compute $(g_1 = x_1x_2, g_2 = x_3x_4)$. Since $\mathcal{M}(\mathcal{K}, M, \psi)$ is assumed to be multiplicative, there exist recombination vectors $\overrightarrow{r}$, $\overrightarrow{t} \in \mathcal{K}^{\sum_{i=1}^{n} {d_i}^2}$, such that

$$x_1^{(1)}x_2^{(1)} =< \overrightarrow{r}, (x_1^{(1)}, x_1^{(2)}, \overrightarrow{\rho_1})M^{\tau} \diamond (x_2^{(1)}, x_2^{(2)}, \overrightarrow{\rho_2})M^{\tau} >, \quad (5)$$

$$x_3^{(2)}x_4^{(2)} =< \overrightarrow{t}, (x_3^{(1)}, x_3^{(2)}, \overrightarrow{\rho_3})M^{\tau} \diamond (x_4^{(1)}, x_4^{(2)}, \overrightarrow{\rho_4})M^{\tau} > . \quad (6)$$

P_j computes $(x_1^{(1)}, x_1^{(2)}, \overrightarrow{\rho_1})(M_{P_j})^{\tau} \diamond (x_2^{(1)}, x_2^{(2)}, \overrightarrow{\rho_2})(M_{P_j})^{\tau} = (\alpha_{j1}, ..., \alpha_{j{d_j}^2}) \in \mathcal{K}^{{d_j}^2}$ and $(x_3^{(1)}, x_3^{(2)}, \overrightarrow{\rho_3})(M_{P_j})^{\tau} \diamond (x_4^{(1)}, x_4^{(2)}, \overrightarrow{\rho_4})(M_{P_j})^{\tau} = (\beta_{j1}, ..., \beta_{j{d_j}^2}) \in \mathcal{K}^{{d_j}^2}$. From (5) and (6) we have

$$x_1^{(1)}x_2^{(1)} = \sum_{j=1}^{n}\sum_{k=1}^{{d_j}^2} r_{jk}\alpha_{jk}, \quad x_3^{(2)}x_4^{(2)} = \sum_{j=1}^{n}\sum_{k=1}^{{d_j}^2} t_{jk}\beta_{jk}. \quad (7)$$

P_j reshares $(\sum_{k=1}^{d_j^2} r_{jk}\alpha_{jk}, \sum_{k=1}^{d_j^2} t_{jk}\beta_{jk})$ by $(\sum_{k=1}^{d_j^2} r_{jk}\alpha_{jk}, \sum_{k=1}^{d_j^2} t_{jk}\beta_{jk}, \overrightarrow{\rho_j}')M^\tau$. Finally, P_j computes

$$\begin{aligned}
&\sum_{i=1}^{n}(\sum_{k=1}^{d_i^2} r_{ik}\alpha_{ik}, \sum_{k=1}^{d_i^2} t_{ik}\beta_{ik}, \overrightarrow{\rho_i}')(M_{P_j})^\tau \\
&= (\sum_{i=1}^{n}\sum_{k=1}^{d_i^2} r_{ik}\alpha_{ik}, \sum_{i=1}^{n}\sum_{k=1}^{d_i^2} t_{ik}\beta_{ik}, \sum_{i=1}^{n} \overrightarrow{\rho_i}')(M_{P_j})^\tau \\
&= (x_1^{(1)}x_2^{(1)}, x_3^{(2)}x_4^{(2)}, \sum_{i=1}^{n} \overrightarrow{\rho_i}')(M_{P_j})^\tau,
\end{aligned}$$

which is P_j's share for $(x_1^{(1)}x_2^{(1)}, x_3^{(2)}x_4^{(2)})$.

Case 3: $(+, \setminus)$ *or* $(\setminus, +)$. Suppose we are to compute $(g_1 = x_1 + x_2, g_2 = x_3)$. Similar to (4), we have $x_3^{(2)} = \sum_{j=1}^{n}\sum_{k=1}^{d_j} b_{jk}s_{3k}^{(j)}$. So each player P_j reshares $(\sum_{k=1}^{d_j} a_{jk}(s_{1k}^{(j)} + s_{2k}^{(j)}), \sum_{k=1}^{d_j} b_{jk}s_{3k}^{(j)})$ through

$$(\sum_{k=1}^{d_j} a_{jk}(s_{1k}^{(j)} + s_{2k}^{(j)}), \sum_{k=1}^{d_j} b_{jk}s_{3k}^{(j)}, \overrightarrow{\rho_j}')M^\tau$$

and finally computes

$$\sum_{i=1}^{n}(\sum_{k=1}^{d_i} a_{ik}(s_{1k}^{(i)}+s_{2k}^{(i)}), \sum_{k=1}^{d_i} b_{ik}s_{3k}^{(i)}, \overrightarrow{\rho_i}')(M_{P_j})^\tau = (x_1^{(1)}+x_2^{(1)}, x_3^{(2)}, \sum \overrightarrow{\rho_i}')(M_{P_j})^\tau,$$

which is P_j's share for $(x_1^{(1)} + x_2^{(1)}, x_3^{(2)})$.

Case 4: $(\times, \setminus)$ *or* $(\setminus, \times)$. It is similar to the above cases and details are omitted here.

Outputting. At the end of computing phase, we can see the final value $(f_1(x_1^{(1)}, ..., x_n^{(1)}), f_2(x_1^{(2)}, ..., x_n^{(2)}))$ is multi-secret shared by using $\mathcal{M}$. If every player is allowed to get the value, in the last phase P_i publics his share for $(f_1(x_1^{(1)}, ..., x_n^{(1)}), f_2(x_1^{(2)}, ..., x_n^{(2)}))$ where $1 \le i \le n$, then every player can compute $(f_1(x_1^{(1)}, ..., x_n^{(1)}), f_2(x_1^{(2)}, ..., x_n^{(2)}))$ by the reconstruction algorithm.

If $f_1(x_1^{(1)}, ..., x_n^{(1)})$ is required to be held only by P_1 and $f_2(x_1^{(2)}, ..., x_n^{(2)})$ is to be held only by P_2, all shares cannot be simply transmitted to P_1 and P_2. Because by doing so, P_1, *resp.* P_2 will also know $f_2(x_1^{(2)}, ..., x_n^{(2)})$, *resp.* $f_1(x_1^{(1)}, ..., x_n^{(1)})$. Fortunately, by the reconstruction algorithm, $f_1(x_1^{(1)}, ..., x_n^{(1)})$ and $f_2(x_1^{(2)}, ..., x_n^{(2)})$ are linear combinations of the shares that all players finally hold, so they can be computed through a simple MPC protocol [9] as follows, while keeping the privacy of the shares thus guaranteeing security for parallel MPC.

Since $(f_1(x_1^{(1)}, ..., x_n^{(1)}), f_2(x_1^{(2)}, ..., x_n^{(2)}))$ is multi-secret shared through $\mathcal{M}$, suppose P_i's share for it is $(s_{i1}, \cdots, s_{id_i}) \in \mathcal{K}^{d_i}$ where $1 \le i \le n$. Similar to the

equality (4), we have that

$$f_1(x_1^{(1)},...,x_n^{(1)}) = \sum_{i=1}^{n}\sum_{k=1}^{d_i} a_{ik}s_{ik}, \quad f_2(x_1^{(2)},...,x_n^{(2)}) = \sum_{i=1}^{n}\sum_{k=1}^{d_i} b_{ik}s_{ik} \ .$$

In order to securely compute $f_1(x_1^{(1)},...,x_n^{(1)})$ such that only P_1 learns the value and other players get nothing new, we need a simple MPC protocol. Precisely, for $1 \le i \le n$, P_i randomly selects $r_{i1}, r_{i2}, \cdots, r_{i(n-1)} \in \mathcal{K}$ and sets $r_{in} = \sum_{k=1}^{d_i} a_{ik}s_{ik} - \sum_{j=1}^{n-1} r_{ij}$. Then P_i secretly transmits r_{ij} to P_j, $1 \le j \le n, j \ne i$. After that P_j locally computes $\lambda_j = \sum_{i=1}^{n} r_{ij}$ and transmits r_j to P_1 where $1 \le j \le n$. The process can be displayed as follows.

$$\begin{array}{lcccccc}
 & & P_1 & \cdots & P_n & & \\
P_1 : \sum_{k=1}^{d_1} a_{1k}s_{1k} & \rightarrow & r_{11} & \cdots & r_{1n} & \sum_{k=1}^{d_1} a_{1k}s_{1k} = \sum_{j=1}^{n} r_{1j} \\
P_2 : \sum_{k=1}^{d_2} a_{2k}s_{2k} & \rightarrow & r_{21} & \cdots & r_{2n} & \sum_{k=1}^{d_2} a_{2k}s_{2k} = \sum_{j=1}^{n} r_{2j} \\
\cdots \quad \cdots & & \cdots & \cdots & \cdots & \cdots \\
P_n : \sum_{k=1}^{d_n} a_{nk}s_{nk} & \rightarrow & r_{n1} & \cdots & r_{nn} & \sum_{k=1}^{d_n} a_{nk}s_{nk} = \sum_{j=1}^{n} r_{nj} \\
 & & \lambda_1 = \sum_{i=1}^{n} r_{i1} & \cdots & \lambda_n = \sum_{i=1}^{n} r_{in} & &
\end{array} \tag{8}$$

Finally, P_1 computes

$$\sum_{j=1}^{n}\lambda_j = \sum_{j=1}^{n}\sum_{i=1}^{n} r_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} r_{ij} = \sum_{i=1}^{n}\sum_{k=1}^{d_i} a_{ik}s_{ik} = f_1(x_1^{(1)},...,x_n^{(1)}) \ .$$

Similarly, $f_2(x_1^{(2)},...,x_n^{(2)})$ can be securely computed and only P_2 gets the final value.

3.3 Comparing with the "Direct Sum" Method

Since the "direct sum" method (in Section 3.1) is a natural way to realize secure parallel multi-party computation, we compare our protocol (in Section 3.2) with it. As to the security issue, note that in our protocol all inputs and outputs for every step is multi-secret shared during the protocol. For any $B \in \mathcal{A}_{i_1} \cap \cdots \cap \mathcal{A}_{i_k}$, it follows that $\{S^{i_1},...,S^{i_k}\} \subseteq \{S^1,...,S^m\} \setminus \{S^i \mid B \in AS_i, 1 \le i \le m\}$. By the security requirement of the LMSSS, players in B get no information about $\{S^{i_1},...,S^{i_k}\}$ from the shares they hold, that is, the intermediate communication data held by players in B tells nothing about other players' inputs for functions $f_{i_1},...,f_{i_k}$. So an adversary corrupting players in B gets no information about other players' (players out of B) inputs for functions $f_{i_1},...,f_{i_k}$, except what can be implied from the inputs and outputs held by players in B. Hence our protocol and the "direct sum" method are of the same security.

The communication complexity is an important criterion to evaluate a protocol. By using a " non-direct sum" LMSSS, our protocol may need less communication than the "direct sum" method, and this advantage becomes more

evident when computing more complicated functions, *i.e.*, the functions essentially contain more variables and more multiplications. In the next section, we show the advantage of communication complexity through a specific example.

4 Example

Suppose that $P = \{P_1, P_2, P_3, P_4, P_5\}$ is the set of players and $|\mathcal{K}| > 5$. Let $AS_1 = \{A \subset P \mid |A| \geq 2 \text{ and } \{P_1, P_2\} \cap A \neq \varnothing\}$ and $AS_2 = \{A \subset P \mid |A| \geq 2 \text{ and } \{P_4, P_5\} \cap A \neq \varnothing\}$ be two access structures over P. The corresponding minimum access structures are as follows:

$$(AS_1)_m = \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_1, P_5\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_2, P_5\}\} ,$$

$$(AS_2)_m = \{\{P_4, P_5\}, \{P_1, P_4\}, \{P_2, P_4\}, \{P_3, P_4\}, \{P_1, P_5\}, \{P_2, P_5\}, \{P_3, P_5\}\} .$$

Obviously, the two corresponding adversary structures $\mathcal{A}_1 = 2^P \setminus AS_1$ and $\mathcal{A}_2 = 2^P \setminus AS_2$ are both Q2. The players are to securely compute multi-functions $f_1 = x_1 + x_2x_3, f_2 = x_1x_2$ against an $(\mathcal{A}_1, \mathcal{A}_2)$-adversary. For $1 \leq i \leq 5$, player P_i has private input $(x_i^{(1)}, x_i^{(2)})$.

By the "direct sum" method, we need to design for f_i a MPC protocol against an $\mathcal{A}_i$-adversary where $1 \leq i \leq 2$. From [3] we know that the key step is to devise LSSS with respect to AS_1 and AS_2, respectively. let

$$M_1 = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} , \quad M_2 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} ,$$

and $\psi_1, \psi_2 : \{1, 2, \cdots, 5\} \to P$ be defined as $\psi_1(i) = \psi_2(i) = P_i$ for $1 \leq i \leq 5$. It is easy to verify that $\mathcal{M}_i(\mathcal{K}, M_i, \psi_i)$ is a multiplicative MSP computing f_{AS_i} with respect to $(1, 0) \in \mathcal{K}^2$ where $1 \leq i \leq 2$. Then the MPC protocol follows. Note that the MPC protocol for computing a single function also has input sharing phase, computing phase and outputting phase.

By the protocol in Sec3.2, first we need to design a LMSSS with respect to the multi-access structure AS_1, AS_2. Let $M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & -2 & -1 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & -1 & -1 \end{pmatrix}$ and $\psi : \{1, 2, ..., 9\} \to P$ be defined as $\psi(1) = \psi(2) = P_1$, $\psi(3) = \psi(4) = P_2$, $\psi(5) = P_3$, $\psi(6) = \psi(7) = P_4$, $\psi(8) = \psi(9) = P_5$. It can be verified that $\mathcal{M}(\mathcal{K}, M, \psi)$ is a MSP

computing f_{AS_1} and f_{AS_2} with respect to the target vectors $\overrightarrow{e_1}, \overrightarrow{e_2}$, and later we are to verify that $\mathcal{M}(\mathcal{K}, M, \psi)$ is multiplicative.

INPUT SHARING. First for $1 \le i \le 3$, P_i multi-secret share his private input $(x_i^{(1)}, x_i^{(2)})$ by randomly choosing $\alpha_i, \beta_i \in \mathcal{K}$ and sending $(x_i^{(1)}, x_i^{(2)}, \alpha_i, \beta_i)(M_{P_j})^\tau$ to player P_j, where $1 \le j \le n$. The following table shows the shares each player holds for $(x_i^{(1)}, x_i^{(2)})$ after the phase.

	$(x_1^{(1)}, x_1^{(2)})$	$(x_2^{(1)}, x_2^{(2)})$	$(x_3^{(1)}, x_3^{(2)})$
P_1	$x_1^{(1)} + \alpha_1 + \beta_1,\ \ \beta_1$	$x_2^{(1)} + \alpha_2 + \beta_2,\ \ \beta_2$	$x_3^{(1)} + \alpha_3 + \beta_3,\ \ \beta_3$
P_2	$\beta_1,\ \ 2x_1^{(1)} + \alpha_1 + \beta_1$	$\beta_2,\ \ 2x_2^{(1)} + \alpha_2 + \beta_2$	$\beta_3,\ \ 2x_3^{(1)} + \alpha_3 + \beta_3$
P_3	$\alpha_1 + \beta_1$	$\alpha_2 + \beta_2$	$\alpha_3 + \beta_3$
P_4	$\alpha_1,\ \ x_1^{(2)} - 2\alpha_1 - \beta_1$	$\alpha_2,\ \ x_2^{(2)} - 2\alpha_2 - \beta_2$	$\alpha_3,\ \ x_3^{(2)} - 2\alpha_3 - \beta_3$
P_5	$2\alpha_1 + \beta_1,\ \ x_1^{(2)} - \alpha_1 - \beta_1$	$2\alpha_2 + \beta_2,\ \ x_2^{(2)} - \alpha_2 - \beta_2$	$2\alpha_3 + \beta_3,\ \ x_3^{(2)} - \alpha_3 - \beta_3$

Denote $(x_i^{(1)}, x_i^{(2)}, \alpha_i, \beta_i)M^\tau = (s_{i1}^{(1)}, s_{i2}^{(1)}, s_{i1}^{(2)}, s_{i2}^{(2)}, s_{i1}^{(3)}, s_{i1}^{(4)}, s_{i2}^{(4)}, s_{i1}^{(5)}, s_{i2}^{(5)})$, that is, P_j holds $s_{ik}^{(j)}$ for $(x_i^{(1)}, x_i^{(2)})$ where $1 \le k \le d_i, 1 \le j \le 5$.

It can be verified that

$$x_1^{(1)} = (x_1^{(1)} + \alpha_1 + \beta_1) - (\alpha_1 + \beta_1), \quad x_1^{(2)} = (\alpha_1 + \beta_1) + \alpha_1 + (x_1^{(2)} - 2\alpha_1 - \beta_1). \tag{9}$$

$$\begin{aligned} x_2^{(1)} x_3^{(1)} = &-(x_2^{(1)} + \alpha_2 + \beta_2)(x_3^{(1)} + \alpha_3 + \beta_3) + \\ &\frac{1}{2}(2x_2^{(1)} + \alpha_2 + \beta_2)(2x_3^{(1)} + \alpha_3 + \beta_3) + \frac{1}{2}(\alpha_2 + \beta_2)(\alpha_3 + \beta_3)\ , \end{aligned} \tag{10}$$

$$\begin{aligned} x_1^{(2)} x_2^{(2)} = &(\alpha_1 + \beta_1)(\alpha_2 + \beta_2) - \alpha_1\alpha_2 + (x_1^{(2)} - 2\alpha_1 - \beta_1)(x_2^{(2)} - 2\alpha_2 - \beta_2) + \\ &(2\alpha_1 + \beta_1)(x_2^{(2)} - \alpha_2 - \beta_2) + (x_1^{(2)} - \alpha_1 - \beta_1)(2\alpha_2 + \beta_2). \end{aligned} \tag{11}$$

The equality (9) gives the reconstruction algorithms for $\{P_1, P_3\}$ to recover $x_1^{(1)}$ and for $\{P_3, P_4\}$ to recover $x_1^{(2)}$, so as in the equality (4), we can set

$$\overrightarrow{a} = (1, 0, 0, 0, -1, 0, 0, 0, 0)\ , \quad \overrightarrow{b} = (0, 0, 0, 0, 1, 1, 1, 0, 0)\ .$$

The equalities (10) and (11) show the MSP $\mathcal{M}(\mathcal{K}, M, \psi)$ is multiplicative. Precisely, if we have

$$\begin{aligned} &(x_1^{(1)}, x_1^{(2)}, \alpha_1, \beta_1)M^\tau \diamond (x_2^{(1)}, x_2^{(2)}, \alpha_2, \beta_2)M^\tau \\ = &(s_{11}^{(1)}s_{21}^{(1)}, s_{11}^{(1)}s_{22}^{(1)}, s_{12}^{(1)}s_{21}^{(1)}, s_{12}^{(1)}s_{22}^{(1)}, s_{11}^{(2)}s_{21}^{(2)}, s_{11}^{(2)}s_{22}^{(2)}, s_{12}^{(2)}s_{21}^{(2)}, s_{12}^{(2)}s_{22}^{(2)}, s_{11}^{(3)}s_{21}^{(3)}, \\ &s_{11}^{(4)}s_{21}^{(4)}, s_{11}^{(4)}s_{22}^{(4)}, s_{12}^{(4)}s_{21}^{(4)}, s_{12}^{(4)}s_{22}^{(4)}, s_{11}^{(5)}s_{21}^{(5)}, s_{11}^{(5)}s_{22}^{(5)}, s_{12}^{(5)}s_{21}^{(5)}, s_{12}^{(5)}s_{22}^{(5)})\ , \end{aligned}$$

then as in the equality (7) the recombination vectors are as follows:

$$\overrightarrow{r} = (-1, 0, 0, 0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, 0, 0, 0, 0, 0, 0, 0, 0)\ ,$$

$$\overrightarrow{t} = (0,0,0,0,0,0,0,0,1,-1,0,0,1,0,1,1,0)\ .$$

We transmit $22\log|\mathcal{K}|$ bits of information in this phase. For simplicity, the functions computed in this example involve a few variables. If all variables are involved in each function, *i.e.*, variables $x_1,...,x_5$ all appear in each function, then we need to transmit $36\log|\mathcal{K}|$ bits in the input sharing phase, while by the "direct sum" method $40\log|\mathcal{K}|$ bits need to be transmitted in this phase.

COMPUTING. This phase consists of two steps.

Step 1: $(\times,\times)$. The output of this step is to be the multi-secret shared $(x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$. From (10) and (11), we can see that in the recombination vector $\overrightarrow{r}$ only P_1, P_2 and P_3 has nonzero coefficients, and in the recombination vector $\overrightarrow{t}$ only P_3, P_4 and P_5 has nonzero coefficients, so P_1 reshares $(u_1,v_1) = (-(x_2^{(1)}+\alpha_2+\beta_2)(x_3^{(1)}+\alpha_3+\beta_3), 0)$, P_2 reshares $(u_2,v_2) = (\frac{1}{2}(2x_2^{(1)}+\alpha_2+\beta_2)(2x_3^{(1)}+\alpha_3+\beta_3), 0)$, P_3 reshares $(u_3,v_3) = (\frac{1}{2}(\alpha_2+\beta_2)(\alpha_3+\beta_3), (\alpha_1+\beta_1)(\alpha_2+\beta_2))$, P_4 reshares $(u_4,v_4) = (0, -\alpha_1\alpha_2+(x_1^{(2)}-2\alpha_1-\beta_1)(x_2^{(2)}-2\alpha_2-\beta_2))$ and P_5 reshares $(u_5,v_5) = (0, (2\alpha_1+\beta_1)(x_2^{(2)}-\alpha_2-\beta_2)+(x_1^{(2)}-\alpha_1-\beta_1)(2\alpha_2+\beta_2))$.

After resharing, as shares of (u_i,v_i), P_1 gets $u_i+\alpha_i'+\beta_i',\ \beta_i'$; P_2 gets $\beta_i',\ 2u_i+\alpha_i'+\beta_i'$; P_3 gets $\alpha_i'+\beta_i'$; P_4 gets $\alpha_i',\ v_i-2\alpha_i'-\beta_i'$ and P_5 gets $2\alpha_i'+\beta_i',\ v_i-\alpha_i'-\beta_i'$, where $1\le i\le 5$. Finally

P_1 computes $\sum_{i=1}^5(u_i+\alpha_i'+\beta_i') = x_2^{(1)}x_3^{(1)}+\sum_{i=1}^5(\alpha_i'+\beta_i')$, and $\sum_{i=1}^5\beta_i'$;

P_2 computes $\sum_{i=1}^5\beta_i'$, and $\sum_{i=1}^5(2u_i+\alpha_i'+\beta_i') = 2x_2^{(1)}x_3^{(1)}+\sum_{i=1}^5(\alpha_i'+\beta_i')$;

P_3 computes $\sum_{i=1}^5(\alpha_i'+\beta_i')$;

P_4 computes $\sum_{i=1}^5\alpha_i'$, and $\sum_{i=1}^5(v_i-2\alpha_i'-\beta_i') = x_1^{(2)}x_2^{(2)}-\sum_{i=1}^5(2\alpha_i'+\beta_i')$;

P_5 computes $\sum_{i=1}^5(2\alpha_i'+\beta_i')$, and $\sum_{i=1}^5(v_i-\alpha_i'-\beta_i') = x_1^{(2)}x_2^{(2)}-\sum_{i=1}^5(\alpha_i'+\beta_i')$.

It can be verified that they are the shares for $(x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$ generated from $M(x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)}, \sum_{i=1}^5\alpha_i', \sum_{i=1}^5\beta_i')^\tau$.

Step 2: $(+,\setminus)$. The output of this step is to be multi-secret shared $(x_1^{(1)}+x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$. Since $(x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$ is multi-secret shared after *Step* 1 and $(x_1^{(1)}, x_1^{(2)})$ is multi-secret shared in the Input Sharing phase, then each player adds his shares for $(x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$ to his shares for $(x_1^{(1)}, x_1^{(2)})$. By the linear combinations given in (9), P_1 reshares $(p_1,q_1) = ((x_1^{(1)}+\alpha_1+\beta_1)+x_2^{(1)}x_3^{(1)}+\sum_{i=1}^5(\alpha_i'+\beta_i'), 0)$, P_3 reshares $(p_3,q_3) = (-(\alpha_1+\beta_1)-\sum_{i=1}^5(\alpha_i'+\beta_i'), \sum_{i=1}^5(\alpha_i'+\beta_i'))$ and P_4 reshares $(p_4,q_4) = (0, \sum_{i=1}^5\alpha_i'+x_1^{(2)}x_2^{(2)}-\sum_{i=1}^5(2\alpha_i'+\beta_i'))$. Finally,

P_1 computes $\sum_{i=1,3,4}(p_i+\alpha_i''+\beta_i'') = x_1^{(1)}+x_2^{(1)}x_3^{(1)}+\sum_{i=1,3,4}(\alpha_i''+\beta_i'')$, and $\sum_{i=1,3,4}\beta_i''$;

P_2 computes $\sum_{i=1,3,4}\beta_i''$, and $\sum_{i=1,3,4}(2p_i+\alpha_i''+\beta_i'') = 2(x_1^{(1)}+x_2^{(1)}x_3^{(1)})+\sum_{i=1,3,4}(\alpha_i''+\beta_i'')$;

P_3 computes $\sum_{i=1,3,4}(\alpha_i'' + \beta_i'')$;

P_4 computes $\sum_{i=1,3,4}\alpha_i''$, and $\sum_{i=1,3,4}(q_i - 2\alpha_i'' - \beta_i'') = x_1^{(2)}x_2^{(2)} - \sum_{i=1,3,4}(2\alpha_i'' + \beta_i'')$;

P_5 computes $\sum_{i=1,3,4}(2\alpha_i'' + \beta_i'')$, and $\sum_{i=1,3,4}(q_i - \alpha_i'' - \beta_i'') = x_1^{(2)}x_2^{(2)} - \sum_{i=1,3,4}(\alpha_i'' + \beta_i'')$.

It can be verified that they are the shares for $(x_1^{(1)} + x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$ generated from $M(x_1^{(1)} + x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)}, \sum_{i=1,3,4}\alpha_i'', \sum_{i=1,3,4}\beta_i'')^\tau$.

In each step dealing with multiplications, our protocol transmits at most $36\log|\mathcal{K}|$ bits of information. By the "direct sum" method, each time we do a multiplication it need to transmit $28\log|\mathcal{K}|$ bits. Assume that f_1 contains p multiplications and f_2 contains q multiplications, where $p \le q$. Then our protocol need transmit $36q\log|\mathcal{K}|$ bits to complete all multiplications, while the "direct sum" method transmits $20(p+q)\log|\mathcal{K}|$ bits. If $p = q$, we see that our protocol transmits $4p\log|\mathcal{K}|$ bits less than the "direct sum" method.

In the last step of this phase, that is, when we do additions, from the reconstruction algorithm given by (9) only P_1, P_3 and P_4 need to reshare their shares. But by the "direct sum" method, no resharing is needed when doing additions. So our protocol transmits at most $22\log|\mathcal{K}|$ bits more than the "direct sum" method when dealing with additions. However, when both functions essentially contain large numbers of multiplications, our protocol has great advantage in communication complexity.

Outputting. Assume that all players are allowed to get the final value of both functions. Then every player publics his share for $(x_1^{(1)} + x_2^{(1)}x_3^{(1)}, x_1^{(2)}x_2^{(2)})$ and can compute the final value by the reconstruction algorithms. If $x_1^{(1)} + x_2^{(1)}x_3^{(1)}$ is assumed to be held by P_1 and $x_1^{(2)}x_2^{(2)}$ is assumed to be held by P_2, then our protocol transmits at most $20\log|\mathcal{K}|$ bits more than the "direct sum" method according to (8). Fortunately, this disadvantage is fixed, that is, it does not depend on the functions we compute.

As a whole, our protocol needs less communication than the "direct sum" method when computing complicated functions.

References

1. Beimel, A.: Secure Schemes for Secret Sharing and Key Distribution. PhD thesis, Technion - Israel Institute of Techonlogy, 1996. Available on Internet http://www.cs.bgu.ac.il/ beimel/pub.html
2. Blundo, C., De Santis, A., Di Crescenzo, G.,: Multi-Secret sharing schemes. Advances in Crypotology-CRYPTO'94. LNCS, Vol. 839, 150-163, 1995.
3. Cramer, R., Damgard, I., Maurer, U.: General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. Proc. EUROCRYPT '00, Springer Verlag LNCS, vol. 1807, pp. 316–334. Full version available from IACR eprint archive, 2000.

4. Ding, C., Laihonen, T., Renvall, A.,: Linear multi-secret sharing schemes and error-correcting codes. Journal of Universal Computer Science, Vol.3, No.9, 1997, 1023–1036
5. Fehr, S.,: Efficient Construction of Dual MSP. manuscript 1999. Available on Internet http://homepages.cwi.nl/ fehr/papers/Feh99.pdf
6. Goldreich, O., Micali, S., Wigderson, A.: How to play ANY mental game.Proceedings of the nineteenth annual ACM conference on Theory of computing, pp.218-229, January 1987, New York, New York, United States.
7. Hirt, M., Maurer, U.,: Player simulation and general adversary structures in perfect multi-party computation. Journal of Cryptology, vol.13, NO. 1, pp.31-60, 2000.
8. Karchmer, M. and Wigderson, A.: On span programs. Proc. 8th Ann. Symp. Structure in complexity Theory, IEEE 1993, pp. 102-111.
9. Xiao, L.,: Secret sharing schemes: theory and application. PhD thesis, Academy of Mathematics and Systems Science, CAS, 2004.
10. Xiao, L., Liu, M.: Linear Multi-secret sharing schemes. Science in China Ser. F Information Sciences, vol. 48, NO.1, pp. 125-136, 2005.
11. Yao, A.: Protocols for Secure Computation. Proc. of IEEE FOGS '82, pp. 160-164, 1982.

Appendix: Construct Multiplicative MSP

Let $\mathcal{M}(\mathcal{K}, M, \psi)$ be a MSP computing f_{AS_1} and f_{AS_2} with respect to $\{\overrightarrow{e_1}, \overrightarrow{e_2}\}$. For simplicity, we use $\overrightarrow{e_1}$, *resp.* $\overrightarrow{e_2}$, to denote vectors with the form $(1, 0, \cdots, 0)$, *resp.* $(0, 1, 0, \cdots, 0)$, without distinguishing the dimensions, and the dimension can be determined from context. From [5] we can assume that the columns of M are linear independent and so $d \geq l$. Compute $\overrightarrow{w_1}, \overrightarrow{w_2}$ be such that $\overrightarrow{w_1}M = \overrightarrow{e_1}$ and $\overrightarrow{w_2}M = \overrightarrow{e_2}$, and compute $\overrightarrow{v_1}, ..., \overrightarrow{v_{d-l}}$ as a basis of the solution space to the linear functions $\overrightarrow{v}M = \overrightarrow{0}$. Then construct a matrix

$$\widetilde{M} = \begin{pmatrix} m_{11} & \cdots & \cdots & m_{1l} & & & \\ \vdots & \ddots & \ddots & \vdots & & & \\ m_{d1} & \cdots & \cdots & m_{dl} & & & \\ \overrightarrow{w_1}^\tau & & & & \overrightarrow{v_1}^\tau & \cdots & \overrightarrow{v_{d-l}}^\tau \\ & \overrightarrow{w_2}^\tau & & & & \overrightarrow{v_1}^\tau & \cdots \ \overrightarrow{v_{d-l}}^\tau \end{pmatrix},$$

where $\begin{pmatrix} m_{11} & \cdots & m_{1l} \\ \vdots & \ddots & \vdots \\ m_{d1} & \cdots & m_{dl} \end{pmatrix} = M$, and the blanks in $\widetilde{M}$ denote zero elements. So $\widetilde{M}$ is a $3d \times (2d - l)$ matrix over $\mathcal{K}$. Define a function $\widetilde{\psi} : \{1, ..., 3d\} \to \{1, ..., n\}$ as follows: For $1 \leq k \leq d$, $\widetilde{\psi}(k) = \psi(k)$; For $d < k \leq 2d$, $\widetilde{\psi}(k) = \psi(k - d)$; For $2d < k \leq 3d$, $\widetilde{\psi}(k) = \psi(k - 2d)$. Therefore we get a MSP $\widetilde{\mathcal{M}}(\mathcal{K}, \widetilde{M}, \widetilde{\psi})$.

Proposition 2. *The monotone span program $\widetilde{\mathcal{M}}(\mathcal{K}, \widetilde{M}, \widetilde{\psi})$ constructed above is a multiplicative MSP computing Boolean functions f_{AS_1} and f_{AS_2} with respect to target vectors $\{\overrightarrow{e_1}, \overrightarrow{e_2}\}$.*

Proof: *Let M_1^*, resp. M_2^* be the matrix composed of rows from the $(d+1)$ th to the $2d$ th row of $\widetilde{M}$, resp. from the $(2d+1)$ th to the $3d$ th row of $\widetilde{M}$. Then M_1^* and M_2^* are two $d \times (2d-l)$ matrices, and $\widetilde{M} = \begin{pmatrix} M0 \\ M_1^* \\ M_2^* \end{pmatrix}$, where $M0$ denotes the $d \times (2d-l)$ matrix generated by adding $2(d-l)$ all zero columns to the right of the original $d \times l$ matrix M. Let $AS_1^* = \{B \subset P \mid \overline{B} \notin AS_1\}$ and $AS_2^* = \{B \subset P \mid \overline{B} \notin AS_2\}$. From [5], the MSP $\mathcal{M}_1^*(\mathcal{K}, M_1^*, \psi)$, resp. $\mathcal{M}_2^*(\mathcal{K}, M_2^*, \psi)$ computes the Boolean function $f_{AS_1^*}$, resp. $f_{AS_2^*}$ with respect to the target vector $\overrightarrow{e_1}$, resp. $\overrightarrow{e_2}$.*

In order to prove that $\widetilde{\mathcal{M}}(\mathcal{K}, \widetilde{M}, \widetilde{\psi})$ computes Boolean functions f_{AS_1} and f_{AS_2} with respect to target vectors $\{\overrightarrow{e_1}, \overrightarrow{e_2}\}$, we need to prove: (1) $\overrightarrow{e_1} \in span\{\widetilde{M}_A\}$ iff $A \in AS_1$; (2) $\overrightarrow{e_2} \in span\{\widetilde{M}_A\}$ iff $A \in AS_2$; (3) If $A \notin AS_1 \cup AS_2$, then $\widetilde{M}$ rejects A with respect to $\{\overrightarrow{e_1}, \overrightarrow{e_2}\}$, ie. $Rank \begin{pmatrix} \widetilde{M}_A \\ \overrightarrow{e_1} \\ \overrightarrow{e_2} \end{pmatrix} = Rank\ \widetilde{M}_A + 2$.

(1)Suppose that $A \in AS_1$. Because $\mathcal{M}(\mathcal{K}, M, \psi)$ computes f_{AS_1} with respect to $\overrightarrow{e_1}$, $\overrightarrow{e_1} \in span\{(M0)_A\} \subset span\{\widetilde{M}_A\}$. On the other hand, suppose that $\overrightarrow{e_1} \in span\{\widetilde{M}_A\}$. If $\overrightarrow{e_1} \in span\{(M0)_A\}$, then $A \in AS_1$ because $\mathcal{M}$ computes f_{AS_1} with respect to $\overrightarrow{e_1}$. Otherwise $(M_1^*)_A$ or $(M_2^*)_A$ must contribute to the generation of $\overrightarrow{e_1}$. If $(M_1^*)_A$ contributes, it is easy to see that its contribution must be $span\{\overrightarrow{e_1}\}$. So $\overrightarrow{e_1} \in span\{(M_1^*)_A\}$. Because $\mathcal{M}_1^*(\mathcal{K}, M_1^*, \psi)$ computes the Boolean function $f_{AS_1^*}$ with respect to the target vector $\overrightarrow{e_1}$, $\overrightarrow{e_1} \in span\{(M_1^*)_A\}$ implies that $A \in AS_1^*$. By the assumption $\mathcal{A}_1 = 2^P - AS_1$ is Q2, $AS_1^* \subset AS_1$ and then $A \in AS_1$. Similarly, if $(M_2^*)_A$ contributes, its contribution must be $span\{\overrightarrow{e_2}\}$. So $\overrightarrow{e_2} \in span\{(M_2^*)_A\}$, and thus $A \in AS_2$. Because $\mathcal{M}(\mathcal{K}, M, \psi)$ computes f_{AS_2} with respect to $\overrightarrow{e_2}$, then $\overrightarrow{e_2} \in span\{M_A\}$. As a result, the contribution of $(M_2^*)_A$ is included in that of $(M0)_A$. Thus we can disregard $(M_2^*)_A$ when generating $\overrightarrow{e_1}$, and we have proved that $\overrightarrow{e_1} \in span\{(M0)_A, (M_2^*)_A\}$ implies $A \in AS_1$.

(2)By the discussion similar to (1), $\overrightarrow{e_2} \in span\{\widetilde{M}_A\}$ iff $A \in AS_2$;

(3)Suppose that $A \notin AS_1 \cup AS_2$. It follows that

$$\begin{aligned} span\{(M0)_A, \overrightarrow{e_1}, \overrightarrow{e_2}\} \cap span\{(M_1^*)_A\} &= span\{(M0)_A, \overrightarrow{e_1}, \overrightarrow{e_2}\} \cap span\{(M_2^*)_A\} \\ &= span\{(M_1^*)_A\} \cap span\{(M_2^*)_A\} = 0\ . \end{aligned} \tag{12}$$

So

$$Rank \begin{pmatrix} \widetilde{M}_A \\ \overrightarrow{e_1} \\ \overrightarrow{e_2} \end{pmatrix} = Rank \begin{pmatrix} (M0)_A \\ \overrightarrow{e_1} \\ \overrightarrow{e_2} \end{pmatrix} + Rank\ (M_1^*)_A + Rank\ (M_2^*)_A \tag{13}$$

$$= Rank\ (M0)_A + 2 + Rank\ (M_1^*)_A + Rank\ (M_2^*)_A \tag{14}$$

$$= Rank\ \widetilde{M}_A + 2\ , \tag{15}$$

where the equality (13) and (15) come from the equality (12), and the equality (14) comes from the fact that $\mathcal{M}$ computes f_{AS_1} and f_{AS_2} with respect to $\{\overrightarrow{e_1}, \overrightarrow{e_2}\}$.

Then we prove that $\widetilde{\mathcal{M}}(\mathcal{K}, \widetilde{M}, \widetilde{\psi})$ is multiplicative. For any $s_1, s_1' \in S^1$, $s_2, s_2' \in S^2$, and $\overrightarrow{\rho}, \overrightarrow{\rho}' \in \mathcal{K}^{2d-l-2}$, denote

$$(s_1, s_2, \overrightarrow{\rho})\widetilde{M}^\tau = (s_1, s_2, \overrightarrow{\rho})((M0)^\tau, (M_1^*)^\tau, (M_2^*)^\tau) = (\overrightarrow{u}, \overrightarrow{v}, \overrightarrow{w}) \ ,$$

where $\overrightarrow{u} = (s_1, s_2, \overrightarrow{\rho})(M0)^\tau \in \mathcal{K}^d$, $\overrightarrow{v} = (s_1, s_2, \overrightarrow{\rho})(M_1^*)^\tau \in \mathcal{K}^d$ and $\overrightarrow{w} = (s_1, s_2, \overrightarrow{\rho})(M_2^*)^\tau \in \mathcal{K}^d$. Then using the operation notations in Section 3.1, we have the following:

$$\begin{aligned} < \overrightarrow{u}, \overrightarrow{v}' >= \overrightarrow{u}\,\overrightarrow{v}'^\tau &= (s_1, s_2, \overrightarrow{\rho})M^\tau M_1^* \begin{pmatrix} s_1' \\ s_2' \\ \overrightarrow{\rho}'^\tau \end{pmatrix} \\ &= (s_1, s_2, \overrightarrow{\rho}) \begin{pmatrix} 1\,0 \cdots 0 \\ 0\,0 \cdots 0 \\ \vdots\ \vdots\ \ddots\ \vdots \\ 0\,0 \cdots 0 \end{pmatrix} \begin{pmatrix} s_1' \\ s_2' \\ \overrightarrow{\rho}'^\tau \end{pmatrix} = s_1 s_1' \ , \end{aligned}$$

$$\begin{aligned} < \overrightarrow{u}, \overrightarrow{w}' >= \overrightarrow{u}\,\overrightarrow{w}'^\tau &= (s_1, s_2, \overrightarrow{\rho})M^\tau M_2^* \begin{pmatrix} s_1' \\ s_2' \\ \overrightarrow{\rho}'^\tau \end{pmatrix} \\ &= (s_1, s_2, \overrightarrow{\rho}) \begin{pmatrix} 0\,0\,0 \cdots 0 \\ 0\,1\,0 \cdots 0 \\ 0\,0\,0 \cdots 0 \\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots \\ 0\,0\,0 \cdots 0 \end{pmatrix} \begin{pmatrix} s_1' \\ s_2' \\ \overrightarrow{\rho}'^\tau \end{pmatrix} = s_2 s_2' \ . \end{aligned}$$

Hence $\widetilde{\mathcal{M}}(\mathcal{K}, \widetilde{M}, \widetilde{\psi})$ is multiplicative.