Secure Multi-Party Matrix Multiplication Based on Strassen-Winograd Algorithm

Jean-Guillaume Dumas, Pascal Lafourcade, Julio Lopez Fenner, David Lucas, Jean-Baptiste Orfila, Clément Pernet, Maxime Puys

Secure Multi-Party Matrix Multiplication Based on Strassen-Winograd Algorithm

Jean-Guillaume Dumas
jean-guillaume.dumas@univ-grenoble-alpes.fr
Université Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, UMR 5224
Grenoble, France

Pascal Lafourcade
pascal.lafourcade@uca.fr
LIMOS, Université Clermont Auvergne
Aubiere, France

Julio Lopez Fenner
julio.lopez@ufrontera.cl
Universidad de La Frontera, Departamento de Ingeneria Matematica
Temuco, Chile

David Lucas
david.lucas@univ-grenoble-alpes.fr
Université Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, UMR 5224
Grenoble, France

Jean-Baptiste Orfila
jean-baptiste.orfila@univ-grenoble-alpes.fr
Université Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, UMR 5224
Grenoble, France

Clément Pernet
clement.pernet@univ-grenoble-alpes.fr
Université Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, UMR 5224
Grenoble, France

Maxime Puys
maxime.puys@univ-grenoble-alpes.fr
Université Grenoble Alpes, Verimag, CNRS
Grenoble, France

ABSTRACT

This paper presents how to adapt Strassen-Winograd matrix multiplication algorithm to the context of secure multiparty computation. We consider that each player owns only one row of both input matrices and learns one row of the product matrix, without revealing their input to other players. After presenting some building block protocols, we describe how to perform a secure execution of Strassen-Winograd. Then, a comparison is made with a variant of the best known algorithm, relaxed to this setting. It first shows that, asymptotically, the communication volume is reduced from $O(n^3)$ to $O(n^{2.81})$, as expected. Furthermore, a finer study on the amount of communication shows that the improvement occurs for matrices with dimension as small as $n = 81$.

1 INTRODUCTION

Secure multiparty computations (MPC) allows n players to compute together the output of some function, using private inputs without revealing them. In the end, the players only know the result and did not learn any other information. As some players can be malevolent in several ways, there exists different setup for the security of such protocols: for instance, resistance against a collusion of many players, or against attackers who modify their input. Several tools exist to design MPC protocols, like Shamir's secret sharing scheme [15], homomorphic encryption [10] or using a Trusted Third Party [6].

Amongst applications of MPC, one can cite the distributed evaluation of trust, as defined in [7, 11]. In this context, players compute confidence by combining their mutual degrees of trust. The aggregation of trust amongst players can be represented as a matrix product $C = A \times B$, where each player knows one row of the matrix containing their partial trust towards their neighbours and the network has to compute a distributed matrix squaring. Hence, in this particular application, it is necessary to be able to efficiently and securely compute matrix-matrix products. Such an algorithm, *YTP-SS*, is described in [8] and uses successive dot products computations. This algorithms runs in $O(n^3)$, in both arithmetic and communication complexity. In this paper, we follow-up on this result by showing that is it possible to achieve a better complexity bound using an MPC version of the Winograd's variant of Strassen's algorithm (referred to *MP-SW*) [16, Alg. 12.1][1, Exercice 6.5 p.247], which decreases the costs to $O(n^{2.81})$. The security level of the algorithm presented here is the same as [8] without additional security measures. By exposing how to apply it in the aforementioned MPC framework, we show that this speed-up carries over for the communication cost. For this, we rely on a partial homomorphic encryption scheme [3] (namely Naccache-Stern protocol [13]) and its ability to perform homomorphic addition and subtraction of matrices, together with additive and multiplicative masking.

As the volume of communications and computations have the same order of magnitude, when trade-offs are possible in terms of constant factor, we here choose to favor communications.

Contrarily to [8], Strassen-Winograd algorithm involves numerous addition and subtraction on parts of the A and B matrices held by distinct players. Security then requires that these entries are encrypted beforehand. As a consequence, the classic matrix multiplication of [8] can no longer be used as is, even as a base case of Strassen-Winograd's algorithm. We therefore propose an alternative base case: its arithmetic cost is higher, but it involves an equivalent amount of communication. Yet, combined with the recursive Strassen-Winograd algorithm it compares favourably to [8] in communication cost for matrices of dimensions larger than $n = 81$.

In Section 2, we recall Strassen-Winograd and *YTP-SS* algorithm, define the data layout and encryption setting and present the automated verification tool used for security proofs. Then, Section 3 describes building block protocols and presents a new cubic-time matrix multiplication algorithm to be used as a base case. Finally, Section 4 describes the new sub-cubic MPC whole Strassen-Winograd algorithm and Section 5 gives a comparative analysis of its communication cost.

2 PRELIMINARIES

2.1 Strassen-Winograd algorithm

The principle of Strassen-Winograd algorithm is to compute the product $C = A \times B$ by splitting the input matrices in four quadrants of equal dimensions: $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$. Each recursive call consists of 22 block operations:

- 8 additions:

$$\begin{array}{lll} S_1 \leftarrow A_{21} + A_{22} & S_2 \leftarrow S_1 - A_{11} & S_3 \leftarrow A_{11} - A_{21} \\ T_1 \leftarrow B_{12} - B_{11} & T_2 \leftarrow B_{22} - T_1 & T_3 \leftarrow B_{22} - B_{12} \\ S_4 \leftarrow A_{12} - S_2 & & T_4 \leftarrow T_2 - B_{21} \end{array}$$

- 7 recursive multiplications:

$$\begin{array}{lll} R_1 \leftarrow A_{11} \times B_{11} & R_2 \leftarrow A_{12} \times B_{21} & \\ R_3 \leftarrow S_4 \times B_{22} & R_4 \leftarrow A_{22} \times T_4 & \\ R_5 \leftarrow S_1 \times T_1 & R_6 \leftarrow S_2 \times T_2 & R_7 \leftarrow S_3 \times T_3 \end{array}$$

- 7 final additions:

$$\begin{array}{lll} U_1 \leftarrow R_1 + R_2 & U_2 \leftarrow R_1 + R_6 & \\ U_3 \leftarrow U_2 + R_7 & U_4 \leftarrow U_2 + R_5 & \\ U_5 \leftarrow U_4 + R_3 & U_6 \leftarrow U_3 - R_4 & U_7 \leftarrow U_3 + R_5 \end{array}$$

- The result is the matrix: $C = \begin{bmatrix} U_1 & U_5 \\ U_6 & U_7 \end{bmatrix}$.

2.2 Data layout and encryption

We consider the setting where the two input matrices A and B of dimension $n \times n$ and each of the n players store one row of A and the corresponding row of B and learns the corresponding row of $C = A \times B$. In this setting, [8, Algorithm 15] manages to compute C by encrypting the rows of A but keeping the rows of B in plaintext: players owning rows of A, encrypt those rows before sending them, then players owning elements of B homomorphically multiply those by their values (the use of Paillier encryption scheme enables one to compute the encryption of a product from one encrypted and one plain multiplicand).

However Strassen algorithm, under consideration here, requires adding and subtracting submatrices of B of distinct row index sets. These operations on non-ciphered rows of B would automatically leak information. We therefore impose that the rows of B are also encrypted following the same distribution as that of A, which is, each player P_k:

(1) holds a row of A and the corresponding row of B,
(2) discovers the corresponding row of the result $C = A \times B$
(3) these rows of A, B and C are encrypted with the public key of another player.

We therefore introduce the notion of location and key sequences for matrix, to identify the roles of the players in this data and encryption layout:

Definition 2.1. An $n \times n$ matrix A, has location sequence $L = (l_1, l_2, \ldots, l_n)$ and key sequence $K = (k_1, k_2, \ldots, k_n)$ if row i of A is stored by player P_{l_i} encrypted with the public key PK_{k_i} of player P_{k_i} for all $1 \leq i \leq n$.

Example 2.2. For $n = 3$, consider the location sequence $L = (2, 3, 1)$ and key sequence $K = (3, 1, 2)$. This means that player P_2 stores row 1 of A encrypted with the public key of player P_3; player P_3 stores row 2 of A encrypted with the public key of player P_1 and finally player P_1 stores row 3 of A encrypted with the public key of player P_2.

In the setting of the matrix multiplication problem to be solved, the initial location sequences of the three operands A, B, C are identical, as well as their key sequences (but not necessarily all along the course of the recursive algorithm, as shown next). We focus on two types of operations: matrix additions or subtractions and matrix multiplications. By the recursive structure of Strassen's algorithm, the latter type is either achieved by Strassen's algorithm or by a classic matrix product algorithm, used as a base case for the recursion. For the sake of simplicity, we consider henceforth that the initial input matrices are of dimension $n \times n$, with $n = m2^k$, so that up to k recursive calls can be made without having to deal with padding with zeroes nor with peeling thin rows or columns.

A recursive step in Strassen-Winograd algorithm splits the input matrices A and B into four quadrants of equal dimensions. Hence the key sequence K and the location sequence L is split into sub-sequences K_U, L_U for the upper half of the rows K_L and L_L for the lower half of the rows. Figure 1 summarizes these notations. Note that the output matrix C is formed by the 4 blocks U_1, U_5, U_6, U_7 defined in Strassen-Winograd's algorithm.

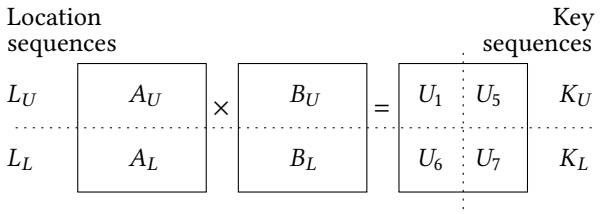

Figure 1: Recursive splitting of the location and key sequences of the input and output operands in Strassen-Winograd's algorithm.

In order to avoid information leakage, we ensure that any intermediate result is stored by a player distinct from the one for which this result is encrypted for. This condition writes:

$$k_i \neq l_i \forall i. \tag{1}$$

Moreover, we impose the following condition, that in any recursive call R_i of Strassen's algorithm, including the calls to base case classical products at the leaves of the recursion tree, the left multiplicand is located and encrypted for a set of players S_A which does not intersect the set of players S_B locating and encrypting the right operand. This implies that:

(1) the base case algorithm (for $m \times m$ matrices) uses $2m$ players,
(2) each recursive call in Strassen's algorithm has to use operands located on non-intersecting set of players.

We propose to use the following values for the location and key sequences, which satisfy all these requirements:

$$\begin{cases} k_i & = & i \text{ for } 0 \leq i < n \\ l_{im+j} & = & k_{im+(j+1 \mod m)} \text{ for } 0 \leq i < n/m, 0 \leq j < m \end{cases}$$

For instance, for a product of dimension 12, with base case dimension $m = 3$, this gives

$$\begin{array}{l} L = (1, 2, 0, 4, 5, 3, 7, 8, 6, 11, 9, 10) \\ K = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) \end{array}.$$

2.3 Homomorphic encryption and initial shift

Notations: for some scalar u and a player A, we denote by $\{u\}_A$ a received message that is the encryption of data u with the public key of A. We also denote $E_A(u)$ the encryption of data u using the public key of A, it means that the player that is generating this cipher-text knows u. We also denote by $r \xleftarrow{\$} D$ the operation of drawing uniformly at random r from a domain D.

Here, our goal is to preserve the inputs privacy during the computation of a matrix multiplication. Hence, the use of homomorphic encryption schemes appears to be natural, since they allow to proceed operations on ciphers. A matrix multiplication is defined over a ring, therefore it requires additive and multiplicative operations over the scalars. On the one hand, we could use a *fully homomorphic* encryption scheme, but this would slow down the protocol too much. On the other hand, we notice that we are dealing with protocols requiring interaction of players. Thus, as described in Section 3, additions and multiplications over ciphers are achievable from a partially homomorphic encryption scheme by using interactive protocols. More precisely, it is sufficient for us that the encryption scheme (G, E, D) satisfies the following properties:

(1) $D_k(E_k(m_1) \times E_k(m_2)) = m_1 + m_2$ (*Additive homomorphism*)
(2) $D_k(E_k(m_1)^{m_2}) = m_1 \times m_2$

Several cryptosystems do satisfy these, e.g., the ones designed by Naccache-Stern [13] or Paillier [14]. On the one hand, in terms of complexity, Naccache-Stern is usually costlier than Paillier: decryption requires to compute small discrete logarithms whereas for Paillier, this is realized with a modular exponentiation. On the other hand, in the context of multiparty protocols, the Naccache-Stern's cryptosystem allows players to agree on a common message block size. This solves the problem of defining a consistent message space between players, which was necessary with the Paillier's cryptosystem. Thus, in the following, we use a Naccache-Stern like cryptosystem.

We use additive masking to protect the privacy of some data, that is replace x by $x + r \mod n$ for $r \xleftarrow{\$} \mathbb{Z}_N$ [9, Proposition 2]. We also sometimes need to use a multiplicative masking. Then, the main difficulty comes from the multiplicative masking of the zero value in a finite field. In our case we use an ad-hoc solution simpler than that of [9]. We consider two matrices $A, B \in \mathbb{Z}_p^{m \times n}$ with p prime and $p > mn$. In order to be able to safely mask B multiplicatively, the players first agree on a public $s \xleftarrow{\$} \mathbb{Z}_p$ and then shift all of their shares of B values by s: $B' = B + s \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \vdots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$. At most mn values for s produce a zero in B', thus the probability that B' has no zero coefficient is greater than $1 - \frac{mn}{p}$. If p has, say, 160 bits and m, n are such that the matrices are storable in an actual computer then this probability is negligible. Finally the protocol can be ran on A and B', multiplicatively masking B', to get $C' = AB'$. To recover $C = AB$, one has just to compute $C = C' - sA \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \vdots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$. If each player owns some rows $A_{\mathcal{I},*}$ of A, then it is easy to locally compute the dotproduct by the vector $s \cdot A_{\mathcal{I},*} \cdot [1, \dots, 1]^T$ and update the obtained shares of C'.

2.4 Automated Verification and attacker models

We use an automatic protocol verification tool to analyse the protocols security. Among existing tools, we use ProVerif [2] which allows us to model protocols using specifically defined equational theories. In our case, we aim to model the encryption scheme properties on ciphertexts. However, as shown in [4], the verification of protocols involving homomorphic functions over abelian groups is undecidable. As a consequence,

we define our own equational theories related to the properties defined in Section 2.3. Moreover, as shown in [12], simple theories such as Exclusive-Or can already exceed the tool's capacities. Thus, the theories we model shall be carefully crafted for our protocols.

In the following, we analyse the protocols presented in Sections 2.5 and 3. For each protocol, we explain the equational theories we introduce to model the used homomorphic encryption primitives. Then, in Section 3.7, we present and describe a table summarizing all results yielded by ProVerif. We first consider a *semi-honest* adversary. Such attacker follows the protocol specifications but tries to learn information from previous computations. In ProVerif, they are modelled as *passive* adversaries, meaning a Dolev-Yao intruder [5] that can only listen to any public communication channel but not tamper with them. We then consider a *malicious* adversary, i.e., an active Dolev-Yao intruder which can freely control a player (this means that the specifications of the protocol might not necessarily be followed).

Our main goal is to prove that in the case where one player is corrupted, there is no leak about the other players' inputs. In other words, we are interested in showing that the protocols respect the **secrecy** property. In ProVerif, this is modelled with the knowledge of the intruder: at the end of the protocol execution, the latter has not learn any of the other player's inputs.

Finally, we assume that players communicate through secure channels, and that a public key infrastructure is available for all the players. This assumption allows us to simplify the proofs. Indeed, since wiretapping a channel does not give any information to the adversary, these data are not added to the adversary view.

2.5 Relaxing an existing algorithm: *YTP-SS*

A recent algorithm on matrix multiplication with the same setup as ours is the secure dot-product protocol *YTP-SS* of [8, Algorithm15], which was used as the main routine in a classical matrix product. As described in the reference above, this protocol is secure against semi-honest adversaries over unsecure communication channels. However, in our setup, channels are assumed secure, hence we can relax *YTP-SS* to make it cheaper in order to compare it fairly with our proposition (*MP-SW*). In this new version, Bob players do not need to protect their inputs with random values. Therefore, the communications performed to derandomize no longer exists. This new version, called here *MP-PDP*, is given in Figure 2.

Theorem 2.3. *Protocol 2 by $n+1$ players requires $2n$ communications. It can be used to compute a classic matrix product by n players, and this product has an overall cost of $n^3 + n(n-1)$.*

Proof. During execution of Protocol 2, Alice first needs to send her value u_i to P_i for $i \in [1, n]$. Then, for $i \in [1, n-1]$, player P_i send their value α_i to P_{i+1}. Finally, P_n sends α_n to Alice, hence the communication volume of $2n$.

First, in the context of matrix multiplication by n player, Alice is one of the other P_i players. Hence only $(n-1)$ communications are required for sharing one row of A and $n(n-1)$ for sharing all of A. Then, the pipelined dot-product phase takes n communication for each coefficient of the output matrix, hence the overall communication volume is $n^3 + n(n-1)$ a classical matrix product with *MP-PDP*. □

2.6 Security Analysis

To model the homomorphic encryption primitives used in Figure 2, we propose the following equational theories. We first define two constructors named *add* and *prod* such that $add(u, v)$ represents $u+v$ and $prod(u, v)$ represents $u \times v$. Then, we propose destructors $uadd1$ and $uadd2$ (resp. $uprod1$ and $uprod2$) to model addition with the opposite of a term (resp. multiplication with the inverse):

(i) $\forall u, v,\ uadd1(add(u, v), v) = u$
(ii) $\forall u, v,\ uadd2(add(u, v), u) = v$

We also propose the following destructor to model homomorphic exponentiation:

(iii) $\forall u, v, k,\ pcr(E_k(u), v) = E_k(prod(u, v))$

Finally, we add the following destructor to model the homomorphic multiplication of two terms:

(iv) $\forall u, v, x, y, k,\quad homth(E_k(prod(u, v)), E_k(prod(x, y))) = E_k(add(prod(u, v), prod(x, y)))$

3 TOOLBOX

3.1 Initialization Phase

Before the actual computation, players need to agree on the location and key sequences they will use and share generate and share their public keys.

Then, Algorithm 1 shows how the input data is ciphered and dispatched between all players: each player gets a row of the matrix indicated by the key sequence and encrypts it with its own public key. It can then send it to the appropriate player hosting the row, according to the location sequence. Algorithm 1 requires $2n^2$ communication.

Algorithm 1 `SWInitialization`

Require: n players $P_1, \ldots, P_n$, P_i owns $a_{i*} = (a_{i1}, \ldots, a_{in})$ and $b_{i*} = (b_{i1}, \ldots, b_{in})$
Require: a sequence $L = (l_1, \ldots, l_n)$ and a sequence $K = (k_1, \ldots, k_n)$
Ensure: Setup is done for *MP-SW* algorithm
  for $i = 1 \ldots n$ **do**
    for $j = 1 \ldots n$ **do**
      P_i computes $\{a_{ij}\}_{P_{k_i}} = E_{P_{k_i}}(a_{ij})$
      P_i computes $\{b_{ij}\}_{P_{k_i}} = E_{P_{k_i}}(b_{ij})$
  for $i = 1 \ldots n$ **do**
    for $j = 1 \ldots n$ **do**
      P_i sends $\{a_{ij}\}_{P_{k_i}}$ to P_{l_i}
      P_i sends $\{b_{ij}\}_{P_{k_i}}$ to P_{l_i}

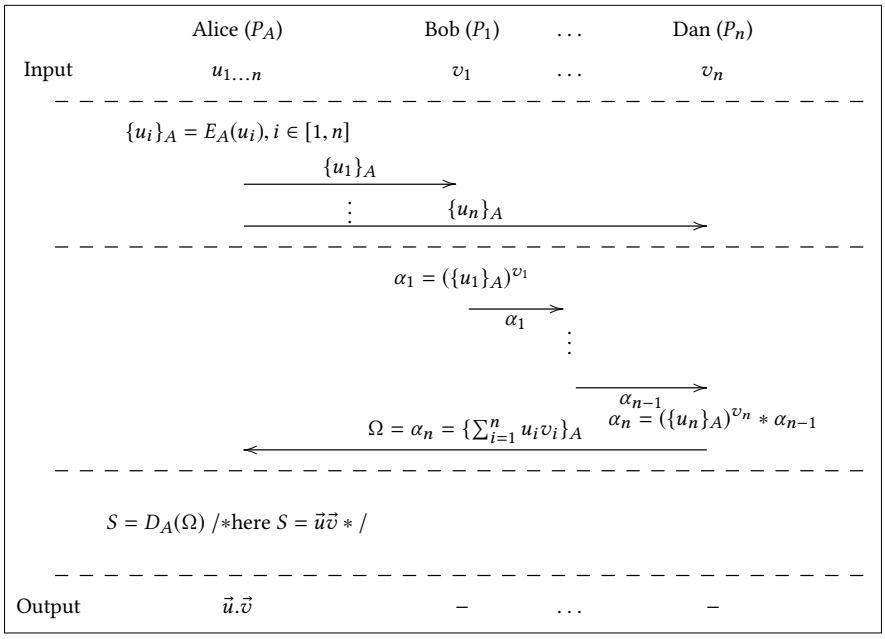

Figure 2: Secure dot-product protocol *MP-PDP*, relaxed from [8, Algorithm 15]

3.2 Multiparty copy

In order to fulfill the specifications of other algorithms, we need to move coefficients from one location to another and change their encryption accordingly.

Figure 3 describes protocol MP-COPY, moving an element x hosted by Bob and encrypted for Dan, to its new location at player Alice and encrypted for player Charlie.

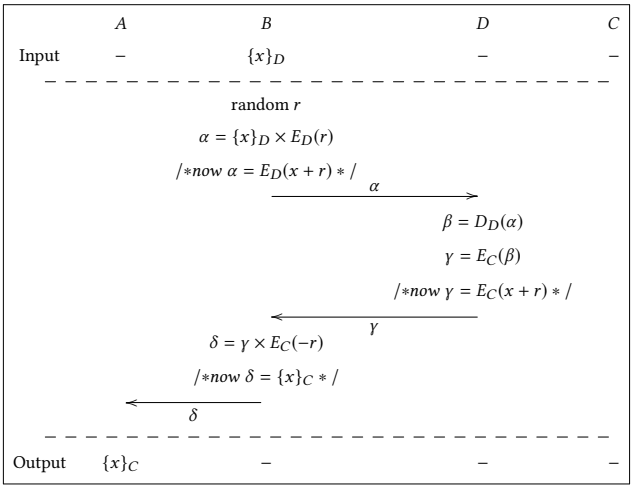

Figure 3: MP-COPY : Multiparty copy

Dan is in charge of performing the decryption and the re-encryption of the element. To prevent him from learning the value of x, Bob masks it additively with a random value. Bob therefore needs to clear out this random mask on the value re-encrypted by Dan, with Charlie's key, before sending it to Alice. This protocol has a total cost of 3 communications.

3.3 Addition and subtraction

When two elements are located at the same player and encrypted for the same other player, addition and subtractions are performed using the homomorphic property of the encryption scheme : $\{x+y\}_C = \{x\}_C \times \{y\}_C$ and $\{x-y\}_C = \{x\}_C / \{y\}_C$.

Now when x and y are located at players A and B and encrypted for players C and D, all four players being distinct, multiparty addition and subtraction, denoted by MP-ADD and MP-SUB, are directly obtained by composing an MP-COPY operation with an homomorphic addition or subtraction. The communication cost for these operations is therefore that of MP-COPY (namely 3 exchanges).

3.4 Security Analysis

To model the homomorphic encryption primitives used in protocol MP-COPY, presented in Figure 3, we propose the following equational theories. We first define two constructors named *add* and *acmask* such that $add(u, v)$ represents $u + v$ and $acmask(E_k(u), E_k(v))$ represents $E_k(u) \times E_k(v)$. Then, we propose an equation describing the inner workings of homomorphic multiplication (resulting in the encrypted sum of terms).

(i) $\forall u, v, k,\ acmask(enc(u, k), enc(v, k)) = enc(add(u, v), k)$

We also provide the following destructors to apply homomorphic addition with opposite of a term:

(ii) $\forall u, v, w, k,\ uacmaskr1(E_k(add(u,v)), E_k(v)) = E_k(u)$
(ii) $\forall u, v, w, k,\ uacmaskr2(E_k(add(u,v)), E_k(u)) = E_k(v)$

Finally, we add the following destructor to model homomorphic decryption:

(iii) $\forall u, v, k,\ decmask(acmask(E_k(u), E_k(u)), k) = add(u,v)$

3.5 Classic Matrix Multiplication base case

We describe in this section an algorithm to perform classic matrix multiplications in the data and encryption layout tailored for its use as a base case for *MP-SW*. In particular, we assume that there are two non intersecting sets of n players each, $P_A = \{A_1 \dots A_n\}$ and $P_B = \{B_1 \dots B_n\}$, such that the location and key sequences for A are permutations of P_A and the location and key sequences for B are permutations of P_B. For the sake of simplicity, we denote in this section by A_i the player in P_A who stored row i of the matrix A encrypted for some other player $D_i \in P_A$. Similarly B_i is the player in P_B who stored row i of the matrix B encrypted for some other player $C_i \in P_B$.

Classic matrix multiplications consist in n^2 scalar products. In each of them, an element $a_{i,k}$ of A is multiplied by an element $b_{k,j}$ of B, with the homomorphic multiplication between a cipher with a key K and a plaintext: $\{a_{i,k}\}_K^{b_{k,j}} = \{a_{i,k}b_{k,j}\}_K$. First, the coefficient $b_{k,j}$ should be deciphered. In order to avoid leaking information, it should be masked by some random value.

Algorithm 2 `MaskAndDecryptB`(k)

Require: $(\mathbf{B}_k)$ knows $\{b_{k,j}\}_{C_k}$ for $j = 1 \dots n$
Ensure: $(\mathbf{C}_k)$ discovers $u_{k,j}$ for $j = 1 \dots n$
  $(\mathbf{B}_k)\ s_k \xleftarrow{\$} \mathcal{F}$
  $(\mathbf{B}_k)$ PRNG.seed(s_k)
  for $j = 1 \dots n$ **do**
    $(\mathbf{B}_k)\ t_{k,j} \leftarrow$ PRNG.next()
    $(\mathbf{B}_k)\ \beta_{k,j} \leftarrow \{b_{k,j}\}_{C_k}^{t_{k,j}}$
    $(\mathbf{B}_k)$ sends $\beta_{k,j}$ to $(\mathbf{C}_k)$
    $(\mathbf{C}_k)\ u_{k,j} \leftarrow D_{C_k}(\beta_{k,j})$

Algorithm 2 takes care of masking and decrypting a whole column of B. There, Player $(\mathbf{C}_k)$ is the only one able to decrypt the masked value $\beta_{k,j} = \{b_{k,j}t_{k,j}\}_{C_k}$. Instead of picking n random values to mask this column, which would yield an overall cubic amount of communication to clear out the mask after the computation, the noise is generated by one random value and n iterations of a deterministic pseudo-random number generator. All players have agreed beforehand on a choice for this pseudo random number generator. Note also that this mask is applied multiplicatively to the value.

Then, Algorithm 3 shows how player $(\mathbf{A}_i)$ discovers the ciphertext of one product $\{a_{i,k}b_{k,j}\}$. Player $(\mathbf{A}_i)$ sends its value $\{a_{i,k}\}$ to player $(\mathbf{C}_k)$ who then performs the exponentiation, corresponding to a multiplication on the plaintexts, and sends

Algorithm 3 `PointwiseProducts`(i,k)

Require: $(\mathbf{A}_i)$ knows $\{a_{i,k}\}_{D_i}$
Require: $(\mathbf{B}_k)$ knows s_k
Require: $(\mathbf{C}_k)$ knows $u_{k,j}$ for $j = 1 \dots n$
Ensure: $(\mathbf{A}_i)$ discovers $\epsilon_{i,k,j} = \{a_{i,k}b_{k,j}\}_{D_i}$ for $j = 1 \dots n$
  $(\mathbf{B}_k)$ sends s_k to $(\mathbf{A}_i)$
  $(\mathbf{A}_i)$ PRNG.seed(s_k)
  $(\mathbf{A}_i)$ sends $\{a_{i,k}\}_{(\mathbf{D}_i)}$ to $(\mathbf{C}_k)$
  for $j = 1 \dots n$ **do**
    $(\mathbf{C}_k)\ \delta_{i,k,j} \leftarrow \{a_{i,k}\}_{D_i}^{u_{k,j}}$
    $(\mathbf{C}_k)$ sends $\delta_{i,k,j}$ to $(\mathbf{A}_i)$
    $(\mathbf{A}_i)\ t_{k,j} \leftarrow$ PRNG.next()
    $(\mathbf{A}_i)\ v_{k,j} \leftarrow t_{k,j}^{-1}$
    $(\mathbf{A}_i)\ \epsilon_{i,k,j} \leftarrow \delta_{i,k,j}^{v_j}$

it back to $(\mathbf{A}_i)$. Meanwhile $(\mathbf{A}_i)$ has received the seed and generated the masking values $t_{k,j}$ to clean out the product.

Algorithm 4 `Reduction`(i)

Require: $(\mathbf{A}_i)$ knows $\epsilon_{i,k,j} = \{a_{i,k}b_{k,j}\}_{D_i}$ for $k, j = 1 \dots n$
Ensure: $(\mathbf{A}_i)$ discovers $\{c_{i,j}\}_{(\mathbf{D}_i)}$ for $j = 1 \dots n$
  $(\mathbf{A}_i)\ \{c_{i,j}\}_{(\mathbf{D}_i)} \leftarrow \prod_{k=1}^{n} \epsilon_{i,k,j}$

Finally each coefficient $\{c_{i,j}\}_{(\mathbf{D}_i)}$ of the result is computed by Algorithm 4 where player $(\mathbf{A}_i)$ simply multiplies all corresponding pointwise products.

Algorithm 5 `BaseCase`

Require: $(\mathbf{A}_i)$ knows $\{a_{i,k}\}_{(\mathbf{D}_i)}$ for $i, k = 1 \dots n$
Require: $(\mathbf{B}_k)$ knows $\{b_{k,j}\}_{(\mathbf{C}_k)}$ for $k, j = 1 \dots n$
Ensure: $(\mathbf{A}_i)$ discovers $\{c_{i,j}\}_{(\mathbf{D}_i)}$ for $i, k = 1 \dots n$
  for $k = 1 \dots n$ **do**
    `MaskAndDecryptB`(k)
    for $i = 1 \dots n$ **do**
      `PointwiseProducts`(i,k)
  for $i = 1 \dots n$ **do**
    `Reduction`(i)

Overall, Algorithm 5 schedules the calls to the three above subroutines. Figure 4 illustrates it in a scenario with 4 players.

Theorem 3.1. *Algorithm 5 computes the product $C = A \times B$ in the data and encryption layout specified. It requires a communication of $n^3 + 2n^2$ integers.*

Proof. The communication cost in number of integer coefficient is n for Algorithm 2 and $n + 1$ for Algorithm 3. Algorithm 3 also sends one seed which size will be neglected in the remaing of the paper. Overall this yields a communication cost of $n^3 + 2n^2$ integers for Algorithm 5. □

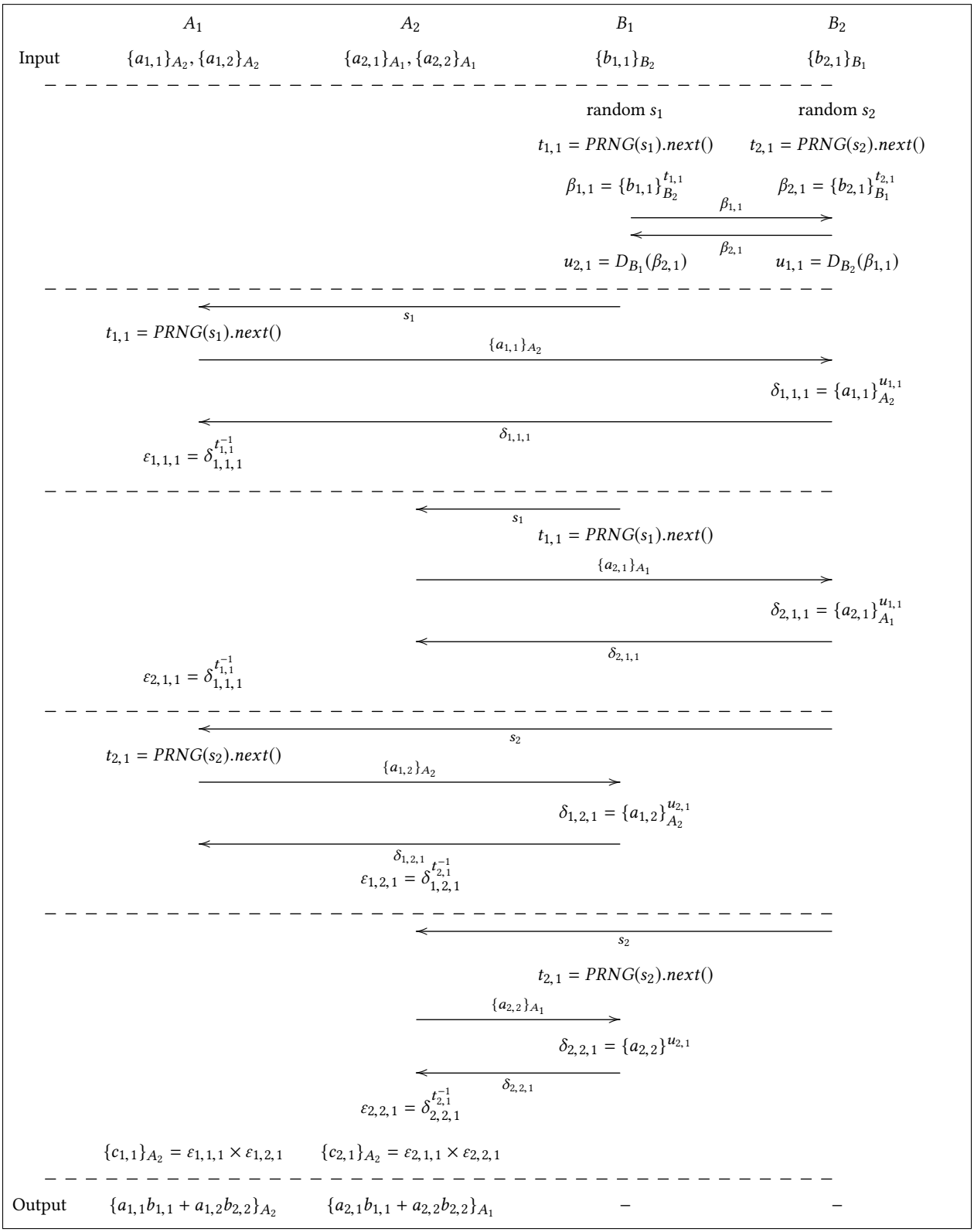

Figure 4: Base-case 4 players

3.6 Security Analysis

To model the homomorphic encryption primitives used in protocol BASE-CASE, presented in Algorithm 5 and instantiated in Figure 4 we propose the following equational theories. We first define the constructors *add* and *acmask* with the exact same properties as in MP-COPY presented in Section 3.4. Then, we propose two new constructors *prng* and *prod* such that $prng(s)$ denotes a pseudo-random number generated from seed s and $prod(x, y)$ denotes $x \times y$. We also propose the following destructor to model homomorphic exponentiation:

(i) $\forall u, v, k,\ pcr(E_k(u), v) = E_k(prod(u, v))$

We add a variant of this destructor where the exponent is a product:

(i) $\forall u, v, w, k,\ pcr2(E_k(u), prod(v, w)) = E_k(prod(u, prod(v, w)))$

We provide a destructor to model homomorphic division:

(i) $\forall u, v, w, k,\ upcr(E_k(prod(u, prod(v, w))), w) = E_k(prod(u, v))$

Finally, we provide the following destructor to model homomorphic multiplicative distributivity:

(ii) $\forall u, v, w, k,\ distri(acmask(E_k(u), E_k(v)), w) = acmask(prod(u, w), prod(v, w))$

3.7 Automated Verification Results

In this section, we summarize all results we found using ProVerif. We considered executability and secrecy properties according to assumptions made in Section 2.4 and with respect to equational theories presented for each protocol. On protocol MP-COPY, we checked secrecy of input x that shall be summed. On protocol BASE-CASE, we checked secrecy of inputs a_{11}, a_{12}, a_{21}, a_{22}, b_{11}, and b_{21}. Finally, on protocol *MP-PDP*, we checked secrecy of inputs u_1, u_2, v_1, and v_2.

Protocols	Active Adversary					Passive Adversary				
	∅	A	B	C	D	∅	A	B	C	D
MP-COPY (Fig. 3)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BASE-CASE (Fig. 4)	✓	?	✓	✓	✓	✓	✓	✓	✓	✓
MP−PDP (Fig. 2)	✓	✓	✓		✓	✓	✓	✓		✓

Table 1: Table of the results obtained with the automated verification tool ProVerif

As we can see in Table 1, all protocols are executable and guarantee the secrecy of inputs for passive intruders (i.e., semi-honest adversary). Out of curiosity, we also considered active intruders. Such adversaries, call Dolev-Yao [5], have a complete control on the network and can intercept, modify, replay and forge any message according to their knowledge. We can see that all protocols are again safe for such adversary except in case of Alice being corrupted for protocol BASE-CASE, where ProVerif does not terminate (denoted by ? in Table 1). This is not surprising since BASE-CASE is a large protocol with consumption-heavy equational theories.

4 MULTIPARTY STRASSEN-WINOGRAD

4.1 Operation schedule in *MP-SW*

The schedule of a recursive step of Strassen-Winograd is composed by 22 operations on the 4 submatrices of each operand. At first, eight block additions, then, seven matrix multiplications are performed, and finally, seven other block additions are computed. Those operations, range in three kind of operations:

• **simple homomorphic addition**: denoted by $+_{\text{HOM}}$. Takes as input two matrices having the same location sequences (and consequently the same key sequences) and returns their sum encrypted with the same key sequence. It is achieved by a simple homomorphic addition. Subtraction is denoted by $-_{\text{HOM}}$ and works similarly.

• **multiparty addition**: denoted by MP-MAT-ADD. Takes as input two matrices having non-intersecting location sequences (and therefore non-intersecting key sequences as well) and returns their sum located and encrypted with location and key sequences of one of the operands. Subtraction is denoted by MP-MAT-SUB. These operations are achieved by n^2 instances of algorithm MP-ADD or MP-SUB, each of which in turn is a composition of MP-COPY and a homomorphic addition or subtraction.

• **multiparty product**: denoted by MP-MAT-MUL. Takes as input two matrices with non-intersecting location sequences (and therefore encrypted with non-intersecting key sequences) and return their product using the location and key sequence of their left-hand side multiplicand. It is achieved by either a recursive call to the same Strassen-Winograd algorithm or by the base case presented in Algorithm 5.

Table 2 presents a schedule of all 22 block operations for a recursive level in Strassen-Winograd algorithm, specifying for each operation the key and location sequences of its operands. It also states which algorithm is used to perform the operation. Figure 5 also presents the data exchange between group of players in one recursive level.

Note that the initial problem requires that both operands A and B share the same key and location sequences (so that matrix squaring is possible). However, the base case algorithm (Algorithm 5) requires that these sequences are non-intersecting. In order to satisfy these two constraints the recursive Strassen-Winograd algorithm (Table 2) is presented with a location and key sequence for A (L_A and K_A) and a location and key sequence for B (L_B and K_B). The algorithm does not require that they are non intersecting, but ensures that from the first recursive call, they will always be, so as to fit with the requirement of Algorithm 5.

Theorem 4.1. *The total communication cost of a recursive level of MP-SW following the schedule defined Table 2 is* $18\left(\frac{n}{2}\right)^2$ *communication.*

Proof. All $+_{\text{HOM}}$ operation are free of communication. Beside recursive calls, there remain 2 calls to MP-MAT-COPY, 2 calls to MP-MAT-ADD and 2 calls to MP-MAT-SUB, each

Operation	Algorithm	Input 1		Input 2		Output	
		Key	Loc.	Key	Loc.	Key	Loc.
$S_1 = A_{21} + A_{22}$	$+_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$A'_{11} = A_{11}$	MP-MAT-COPY	K_{A_U}	L_{A_U}			K_{A_L}	L_{A_L}
$S_2 = S_1 - A'_{11}$	$-_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$S_3 = A'_{11} - A_{21}$	$-_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$S_4 = A_{12} - S_2$	MP-MAT-SUB	K_{A_U}	L_{A_U}	K_{A_L}	L_{A_L}	K_{A_U}	L_{A_U}
$T_1 = B_{12} - B_{11}$	$-_{\text{HOM}}$	K_{B_U}	L_{B_U}	K_{B_U}	L_{B_U}	K_{B_U}	L_{B_U}
$B'_{22} = B_{22}$	MP-MAT-COPY	K_{B_L}	L_{B_L}			K_{B_U}	L_{B_U}
$T_2 = B'_{22} - T_1$	$-_{\text{HOM}}$	K_{B_U}	L_{B_U}	K_{B_U}	L_{B_U}	K_{B_U}	L_{B_U}
$T_3 = B'_{22} - B_{12}$	$-_{\text{HOM}}$	K_{B_U}	L_{B_U}	K_{B_U}	L_{B_U}	K_{B_U}	L_{B_U}
$T_4 = T_2 - B_{21}$	MP-MAT-SUB	K_{B_U}	L_{B_U}	K_{B_L}	L_{B_L}	K_{B_U}	L_{B_U}
$R_1 = A'_{11} \times B_{11}$	MP-MAT-MUL	K_{A_L}	L_{A_L}	K_{B_U}	L_{B_U}	K_{A_L}	L_{A_L}
$R_2 = A_{12} \times B_{21}$	MP-MAT-MUL	K_{A_U}	L_{A_U}	K_{B_L}	L_{B_L}	K_{A_U}	L_{A_U}
$R_3 = S_4 \times B_{22}$	MP-MAT-MUL	K_{A_U}	L_{A_U}	K_{B_L}	L_{B_L}	K_{A_U}	L_{A_U}
$R_4 = A_{22} \times T_4$	MP-MAT-MUL	K_{A_L}	L_{A_L}	K_{B_U}	L_{B_U}	K_{A_L}	L_{A_L}
$R_5 = S_1 \times T_1$	MP-MAT-MUL	K_{A_L}	L_{A_L}	K_{B_U}	L_{B_U}	K_{A_L}	L_{A_L}
$R_6 = S_2 \times T_2$	MP-MAT-MUL	K_{A_L}	L_{A_L}	K_{B_U}	L_{B_U}	K_{A_L}	L_{A_L}
$R_7 = S_3 \times T_3$	MP-MAT-MUL	K_{A_L}	L_{A_L}	K_{B_U}	L_{B_U}	K_{A_L}	L_{A_L}
$U_1 = R_1 + R_2$	MP-MAT-ADD	K_{A_L}	L_{A_L}	L_{A_U}	L_{A_U}	K_{A_U}	L_{A_U}
$U_2 = R_1 + R_6$	$+_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$U_3 = U_2 + R_7$	$+_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$U_4 = U_2 + R_5$	$+_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$U_5 = U_4 + R_3$	MP-MAT-ADD	K_{A_L}	L_{A_L}	K_{A_U}	L_{A_U}	K_{A_U}	L_{A_U}
$U_6 = U_3 - R_4$	$-_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}
$U_7 = U_3 + R_5$	$+_{\text{HOM}}$	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}	K_{A_L}	L_{A_L}

Table 2: Schedule of the secure multiparty Strassen-Winograd algorithm.

accounting for $3(n/2)^2$ communication. Overall, the communication cost of one recursive level is therefore $18\left(\frac{n}{2}\right)^2$. □

4.2 Cost Analysis

The recurrence relation for communication complexity of *MP-SW* writes:

$$\begin{cases} C(n) & = & 7C\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \text{ for } n > m \\ C(m) & = & m^3 + 2m^2 \end{cases}$$

The threshold at which the recursive algorithm should switch to the base case algorithm is set by finding at which dimension does the base case algorithm start to perform worse than one recursive level. In terms of communication cost, this means the following equation: $7((\frac{n}{2})^3 + 2(\frac{n}{2})^2) + 18(\frac{n}{2})^2 = n^3 + 2n^2$ which comes from injecting the base case cost into the recurrence formula. It gives a threshold of $n = 48$.

We now compare the cost of *MP-SW* with the cost of *MP-PDP*, which is $C_{\text{MP-PDP}}(n) = n^3 + n(n-1)$. We also recall that we have the cost of the initialization step of *MP-SW*, $C_{\text{init}} = 2n^2$ and the cost of the final step of *MP-SW*, $C_{\text{final}} = n^2$. In order to find the matrix size for which *MP-SW* has a lower communication cost than *MP-PDP*, we solve the following equation: $C(n) + 3n^2 \leq n^3 + n(n-1)$ which yields $n > 80$, with one recursive call. This means that for any instance of dimension larger than 80, the proposed *MP-SW* algorithm has a better communication cost than *MP-PDP*.

5 EXPERIMENTS

We implemented the algorithms under study[1] to demonstrate their behaviour in practice and compare it to the state of the art implementation of [8]. In the following *YTP-SS* refers to n^2 applications of [8, Algorithm 15]; *MP-PDP* refers the relaxation and improvement of this algorithm to the current setting, as proposed in Figure 2; *MP–SW* refers to our implementation of Algorithm 2 using Algorithm 5 as a basecase with threshold set to $n = 48$. The Naccache-Stern cryptosystem is set with public keys of size 2048 bits and message space of 224 bits (using 14 primes of 16 bits).

Figure 6 presents the volume of communication performed by these three variants. Note that the cross-over point of $n = 80$ is confirmed experimentally. The improvement in communication volume is about 27.8% for $n = 528$ players between algorithms *MP-SW* and *MP-PDP*.

However the running time of Naccache-Stern decryption remains the main bottleneck. Table 3 compares the computation time per player of our implementation of *MP-SW* versus

[1] C++ source files are available on request via the PC chair and will be made publicly available if the paper is accepted.

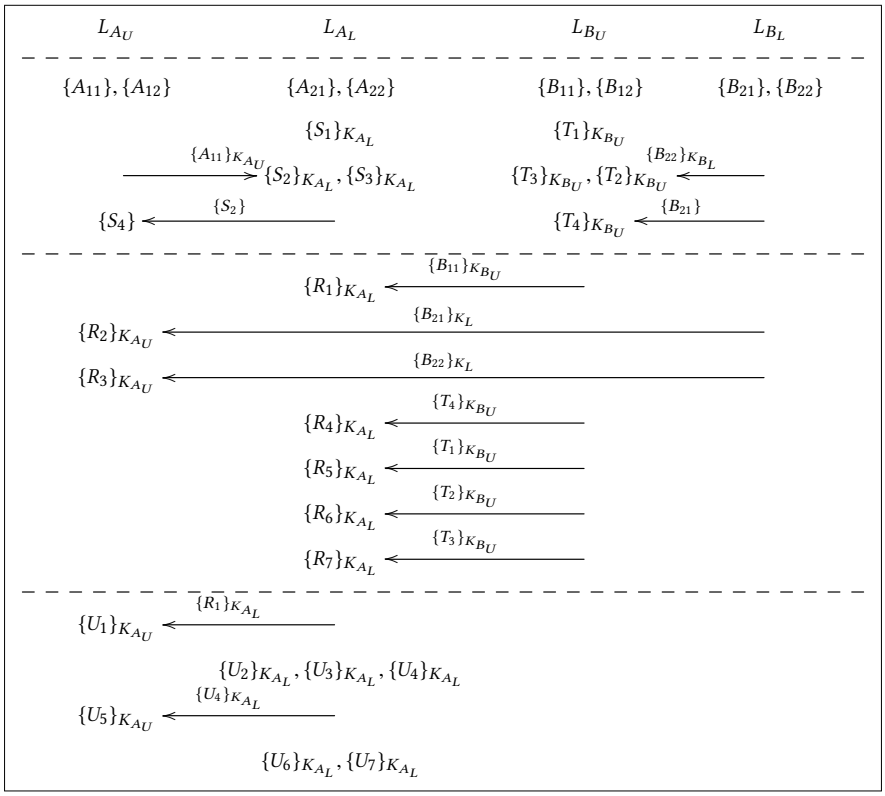

Figure 5: Protocol for Strassen-Winograd algorithm. Each column represents one of the four sub-group of players where the submatrices A_U, A_L, B_U, B_L are stored.

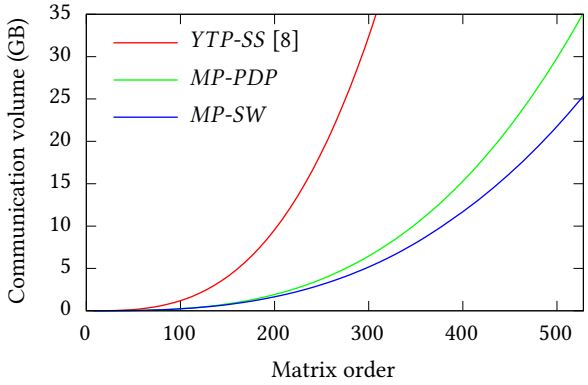

Figure 6: Communication volume for multiparty matrix multiplication.

the implementation of *YTP-SS* provided in [8]. For instance, it takes about 49.22s per player for the *MP-SW* protocol with matrices of size $n = 32$ and keys of 2048 bits, instead of 18.05s for *YTP-SS*.

key size	$n \times n$:	16	32	64
1024	*YTP-SS*	0.58	2.68	11.01
1024	*MP-SW*	2.87	6.19	13.27
2048	*YTP-SS*	4.54	18.05	69.80
2048	*MP-SW*	23.63	49.22	196.24

Table 3: Computation time (in s) per player of Multiparty Strassen-Winograd (*MP-SW*) compared to [8] (*YTP-SS*) on an Intel Xeon E7-8860 2.2Ghz.

6 PERSPECTIVE

The version of Strassen-Winograd we presented here is secure against semi-honest adversaries. However, as many of its building blocks have a stronger security level anyway, it would be interesting to see if it is possible to increase the security of the whole *MP-SW* protocol and how it would impact its performance.

Even if this paper is a about improving the communication cost while preserving security, several arithmetic cost could be envisioned. For instance, removing the need for players to encrypt their own B_k data beforehand. This is required for now to preserve the security features but a large part of the

computing cost lies in the part when one has to decipher and re-cipher that data. Another possibility would be to replace the Naccache-Stern by a faster cryptosystem. The difficulty there is to be able to combine the masking schemes with the somewhat homomorphic properties.

REFERENCES

[1] A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Boston, MA, USA, 1974.

[2] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop, CSFW-14, 11-13 June 2001, Cape Breton, NS, Canada*, pages 82–96, 2001. doi:10.1109/CSFW.2001.930138.

[3] R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge U. Press, New York, NY, USA, 2015.

[4] S. Delaune. An undecidability result for AGh. *Theor. Comput. Sci.*, 368(1-2):161–167, Dec. 2006. doi:10.1016/j.tcs.2006.08.018.

[5] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[6] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *Proceedings of the 2002 Workshop on New Security Paradigms*, NSPW '02, pages 127–135, New York, NY, USA, 2002. ACM. doi:10.1145/844102.844125.

[7] J.-G. Dumas and H. Hossayni. Matrix powers algorithms for trust evaluation in public-key infrastructures. In A. Jøsang, P. Samarati, and M. Petrocchi, editors, *Security and Trust Management*, pages 129–144, Berlin, Heidelberg, 2013. Springer.

[8] J.-G. Dumas, P. Lafourcade, J.-B. Orfila, and M. Puys. Dual protocols for private multi-party matrix multiplication and trust computations. *Computers & Security*, 71:51–70, 2017. URL: http://hal.archives-ouvertes.fr/hal-01344750, doi:10.1016/j.cose.2017.04.013.

[9] L. Genelle, E. Prouff, and M. Quisquater. Secure multiplicative masking of power functions. In J. Zhou and M. Yung, editors, *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, volume 6123 of *LNCS*, pages 200–217, 2010. doi:10.1007/978-3-642-13708-2_13.

[10] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In C.-s. Park and S. Chee, editors, *Information Security and Cryptology – ICISC 2004*, pages 104–120, Berlin, Heidelberg, 2005. Springer.

[11] A. Josang. Probabilistic logic under uncertainty. In J. Gudmundsson and B. Jay, editors, *13th Computing: Australasian Theory Symposium (CATS2007)*, volume 65 of *CRPIT*, pages 101–110, Ballarat, Australia, 2007. ACS.

[12] P. Lafourcade and M. Puys. Performance evaluations of cryptographic protocols verification tools dealing with algebraic properties. In *Foundations and Practice of Security, FPS 2015, Clermont-Ferrand, France, October 26-28, 2015, Revised Selected Papers*, pages 137–155, 2015. URL: http://maxime.puys.name/publications/pdf/LP15.pdf, doi:10.1007/978-3-319-30303-1_9.

[13] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *Proc. of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 59–66, New York, NY, USA, 1998. ACM. doi:10.1145/288090.288106.

[14] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999. doi:10.1007/3-540-48910-X_16.

[15] A. Shamir. How to share a secret. *Comm. ACM*, 22(11):612–613, Nov. 1979. doi:10.1145/359168.359176.

[16] G. J. von zur Gathen J. *Modern computer algebra*. Cambridge University Press, 3ed. edition, 2013.