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

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

Secure Multiparty Matrix Multiplication Based on Strassen-Winograd Algorithm *

Jean-Guillaume Dumas[1], Pascal Lafourcade[2], Julio Lopez Fenner[3], David Lucas[1], Jean-Baptiste Orfila[1], Clément Pernet[1], and Maxime Puys[1]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP**, LJK, VERIMAG, 38000 Grenoble, France. fistname.lastname@univ-grenoble-alpes.fr
[2] LIMOS, Université Clermont Auvergne. 1, rue de Chebarde, 63178 Aubière. France.pascal.lafourcade@uca.fr
[3] Universidad de La Frontera, Departamento De Ingenieria Matematica. Av. Francisco Salazar 01145, Temuco, Chile.julio.lopez@ufrontera.cl

Abstract. This paper presents a secure multiparty computation protocol for the Strassen-Winograd matrix multiplication algorithm. We focus on the setting in which any given player knows only one row (or one block of rows) of both input matrices and learns the corresponding row (or block of rows) of the resulting product matrix. Neither the player initial data, nor the intermediate values, even during the recurrence part of the algorithm, are ever revealed to other players. We use a combination of partial homomorphic encryption schemes and additive masking techniques together with a novel schedule for the location and encryption layout of all intermediate computations to preserve privacy. Compared to state of the art protocols, the asymptotic communication volume of our construction is reduced from $O(n^3)$ to $O(n^{2.81})$. This improvement in terms of communication volume arises with matrices of dimension as small as $n = 96$ which is confirmed by experiments.

1 Introduction

Secure multiparty computations (MPC) allows n players to compute together the output of some function, using private inputs without revealing them. This is useful, e.g., for a distributed evaluation of trust, as defined in [17,10]. In this context, players compute a confidence level by combining their mutual degrees of trust. This aggregation of trust among 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 neighbors and the network has to compute a distributed matrix exponentiation, which reduces to several matrix multiplications. In this paper we thus focus on this particular layout of data, and on multiparty matrix multiplication of dimension $N \times N$ with N players.

* This research was partly supported by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

** Institute of Engineering, Univ. Grenoble Alpes

Several tools exist to design MPC protocols, like Shamir's secret sharing scheme [27], homomorphic encryption [13], oblivious transfer [5] or using a Trusted Third Party [8]. Then, several MPC implementations are available[4]. Some of them are for two parties only and most of the others are generic and transform programs into circuits or use oblivious transfer [7,26,6,16,23]. For instance the symmetric system solving phase of the LINREG-MPC software is reported in [12] to take about 45 minutes for $n = 200$, while, in [11], a secure multiparty specific algorithm, `YTP-SS`, was developed for matrix multiplication requires about a hundred seconds to perform an $n = 200$ matrix multiplication. These timings, however, do not take into account communications, but for multiparty matrix multiplication, the number of communications and the number of operations should be within the same order of magnitude. Our goal is thus to improve on existing algorithms, primarily in terms of this number of communications (we do not minimize the number of messages, as in [15], but instead consider the overall volume). Our idea is to use an algorithm with a lower time and communication complexity for matrix multiplication. Strassen's algorithm [28] was the first sub-cubic time algorithm, with an exponent $\log_2 7 \approx 2.81$, with a complexity of $O(n^{2.81})$ and we hence construct an MPC protocol based Winograd's variant of this algorithm[5] [1, Ex.6.5].

To preserve the inputs privacy during the computation of a matrix multiplication, the use of homomorphic encryption schemes appears to be natural. While we could use a fully homomorphic encryption scheme, it would slow down the protocol unreasonably. Instead, we will use partial homomorphic encryption scheme [3] as they allow to perform the operations we need, namely:

1. $D_{sk}(E_{pk}(m_1) \times E_{pk}(m_2)) = m_1 + m_2$ (*Additive homomorphism*)
2. $D_{sk}(E_{pk}(m_1)^{m_2}) = m_1 \times m_2$ (*Cipher/clear multiplicative homomorphism*)

Several cryptosystems do satisfy these, e.g., the ones designed by Naccache-Stern or Paillier [24,25]. The former is usually costlier than the latter. However, as the former allow parties to agree on a common message block size, which solves the issue of defining a consistent message space among them, we choose here to use the Naccache-Stern cryptosystem.

Finally, Strassen-Winograd algorithm involves numerous additions and subtractions on parts of the A and B matrices that are held by different players. Security concerns require then that these entries should be encrypted from the start, contrarily to [11]. As a consequence, the classical matrix multiplication can no longer be used as stated in the latter reference, even for the base case of the recursive algorithm. We therefore propose an alternative base case. Its arithmetic cost is higher, but it involves an equivalent amount of communication. We shall show that this choice combined with our multiparty recursive

[4] http://www.multipartycomputation.com/mpc-software

[5] The best value known to date, due to LeGall's [21], of approximately 2.3728639. However, only a few sub-cubic time algorithms are competitive in practice and used in software [9,2,19] (see also [20] and references therein), among which Strassen's algorithm and its variants stand out as a very effective one in practice.

Strassen-Winograd algorithm compares favorably to existing implementations in communication cost for matrices of dimensions larger than $N = 96$.

Hypotheses. In this paper, we will only consider the case of *semi-honest* (also called *honest-but-curious*) adversaries. Such adversaries, represented as probabilistic polynomial time machines, try to gather as many information as possible during the execution of the protocol, and can locally run any computation based on this information in order to deduce some private input. However, they strictly follow protocol specifications. We also consider that *communications are performed over secure channels*: this means transferred data is resistant to eavesdropping and that only the recipient will learn anything from communicated data.

Contributions. We propose an instance of Strassen-Winograd's algorithm in a secure multiparty computation setting where the input and output matrices are split and shared row-wise. More precisely, this paper presents the following contributions:

1. A schedule of the operations of Strassen-Winograd's algorithm and of a classic matrix multiplication algorithm compliant with a privacy-preserving location and encryption data-layout;
2. A recursive protocol proven secure against one semi-honest adversary;
3. A reduction of the overall amount of communication from $O(N^3)$ to $O(N^{2.801})$ for the multiparty multiplication of $N \times N$ matrices;
4. This improvement is confirmed by experiments showing advantages of this approach over alternative implementations of MPC matrix multiplication protocols.

The article proceeds as follows: Section 2 presents Strassen-Winograd and the concurrent `YTP-SS` algorithms. There, we also define the dedicated data layout and the cryptographic tools we will use. Next, in Section 3, we first describes our building block protocols, with their security analysis. Second, we present in this Section a new cubic-time matrix multiplication algorithm on ciphered entries to be used as a base case. Section 4 describes the complete novel sub-cubic MPC Strassen-Winograd algorithm and details its theoretical communication cost. Finally, Section 5 closes with practical comparisons between our C++ and concurrent implementations.

2 Preliminaries

2.1 Strassen-Winograd algorithm

$C = A \times B$ by splitting the input matrices in four quadrants of equal dimensions: $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right]$ and $B = \left[\begin{smallmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{smallmatrix}\right]$. Each recursive call consists in 22 block operations:

– 8 additions:]

$$S_1 \leftarrow A_{21} + A_{22} \quad S_2 \leftarrow S_1 - A_{11} \quad S_3 \leftarrow A_{11} - A_{21} \quad S_4 \leftarrow A_{12} - S_2$$
$$T_1 \leftarrow B_{12} - B_{11} \quad T_2 \leftarrow B_{22} - T_1 \quad T_3 \leftarrow B_{22} - B_{12} \quad T_4 \leftarrow T_2 - B_{21}$$

– 7 recursive multiplications:

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

– 7 final additions:

$U_1 \leftarrow R_1 + R_2$ $U_2 \leftarrow R_1 + R_6$ $U_3 \leftarrow U_2 + R_7$ $U_4 \leftarrow U_2 + R_5$
$U_5 \leftarrow U_4 + R_3$ $U_6 \leftarrow U_3 - R_4$ $U_7 \leftarrow U_3 + R_5$

– The result is the matrix: $C = \left[\begin{smallmatrix} U_1 & U_5 \\ U_6 & U_7 \end{smallmatrix}\right]$.

Although the recursion could be run down to products of 1×1 matrices, it is commonly stopped at a fixed dimension threshold, where a classical cubic time algorithm is then used, in order to reduce the overhead of recursion on small dimension instances. For the sake of simplicity, we consider henceforth that the initial input matrices are of dimension $N \times N$, with $N = b2^\ell$, so that up to ℓ recursive calls can be made without having to deal with padding with zeroes nor with peeling thin rows or columns.

2.2 Data layout and encryption

We consider the setting where the two input matrices A and B have dimension $N \times N$ and each of the N players stores one row of A and the corresponding row of B and learns the corresponding row of $C = A \times B$. In this setting, the `YTP-SS` Algorithm [11, Algorithm 15] can compute C by encrypting the rows of A only and then relying on homomorphic multiplications of encrypted coefficients of A by plain coefficients of B.

However, Strassen's algorithm, considered here, requires adding and subtracting submatrices of B of distinct row index sets (e.g. $T_3 \leftarrow B_{22} - B_{12}$). These operations on non-ciphered rows of B would automatically leak information. We therefore impose that the rows of both operands A and B, of the result C and of any intermediate matrix are encrypted by the public key of a player who is not the one hosting the row. We therefore introduce the notion of location and key sequences for a matrix, to identify the roles of the players in this data layout:

Definition 1. *An $n \times n$ matrix A of* ciphered values *has location sequence $L = (l_1, l_2, \ldots, l_n)$ and key sequence $K = (k_1, k_2, \ldots, k_n)$ if player P_{l_i} stores row i of A, that was encrypted with the public key pk_{k_i} of player P_{k_i} for all $1 \leq i \leq n$.*

Example 1. 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 matrix multiplication algorithms presented in the later sections, the location and key sequences of operand A and C will always be identical. On the other hand the location and key sequences of B may equal those of A (in the

first recursive call), or differ but they must then have an empty intersection with those of A.

A recursive step in Strassen-Winograd algorithm splits the matrices A, B and C into four quadrants of equal dimensions. Hence their key and location sequences are split into two sub-sequences: for $X \in \{A, B, C\}$, $L_X = (L_{X_U}, L_{X_L})$ and $K_X = (K_{X_U}, K_{X_L})$ such that (L_{X_U}, K_{X_U}) are the location and key sequences for the upper half of X and (L_{X_L}, K_{X_L}) are the location and key sequences for the lower half of X.

Figure 1 summarizes these notations.

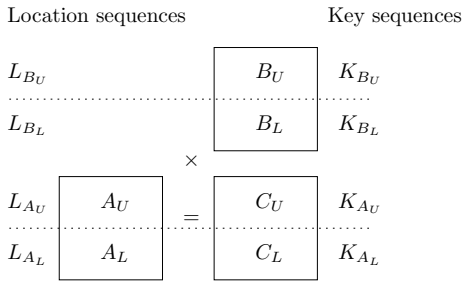

Fig. 1. Recursive splitting of the location and key sequences of the input and output operands in Strassen-Winograd algorithm.

More formally, we present in Definition 2 the two distinct data layouts used in our algorithms: one for the recursive levels of Strassen-Winograd, and one for its base case.

Definition 2. *Let $N \in \mathbb{N}$, $n \leq N$ and A and B two $n \times n$ matrices with location and key sequences $(L_A, K_A) \in (\{1..N\}^n)^2$ and $(L_B, K_B) \in (\{1..N\}^n)^2$.*

1. *(L_A, K_A, L_B, K_B) is a valid data layout if*
   (a) *$\forall i \in \{1..n\}$, $L_A[i] \neq K_A[i]$ and $L_B[i] \neq K_B[i]$.*
   (b) *$\forall i, j \in \{1..n\}$ with $i \neq j$, $L_A[i] \neq L_A[j]$ and $L_B[i] \neq L_B[j]$*
   (c) *$\forall i, j \in \{1..n\}$ with $i \neq j$, $K_A[i] \neq K_A[j]$ and $K_B[i] \neq K_B[j]$*
2. *(L_A, K_A, L_B, K_B) is a base case or a 0-recursive data layout if it is a valid data layout and $(L_A \cup K_A) \cap (L_B \cup K_B) = \emptyset$.*
3. *(L_A, K_A, L_B, K_B) is a ℓ-recursive data layout if it is a valid data layout and*
   (a) *$(L_{A_U} \cup K_{A_U}) \cap (L_{A_L} \cup K_{A_L}) = \emptyset = (L_{B_U} \cup K_{B_U}) \cap (L_{B_L} \cup K_{B_L})$*
   (b) *$(L_{A_U}, K_{A_U}, L_{B_L}, K_{B_L})$ and $(L_{A_L}, K_{A_L}, L_{B_U}, K_{B_U})$ are both $(\ell-1)$-recursive data layouts*

For $N = b2^\ell$, we propose to use the following values for the location and key sequences, to form an ℓ-recursive data layout:

$$\begin{cases} k_i = i & \text{for } 0 \leq i < N \\ l_{ib+j} = ib + (j+1 \mod b) & \text{for } 0 \leq i < N/b, \text{and } 0 \leq j < b \end{cases} \tag{1}$$

For instance, for a product of dimension 12, with base case dimension $b = 3$, this gives; $L_A = L_B = L_C = (1, 2, 0, 4, 5, 3, 7, 8, 6, 11, 9, 10)$ and $K_A = K_B = K_C = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$.

2.3 Homomorphic encryption

Naccache-Stern cryptosystem. In the following, we use Naccache-Stern [24] partially homomorphic cryptosystem, with security parameter 1^λ, set up as follows:

Setup(1^λ) : Select $2k$ small primes $p_1, \ldots, p_{2k}$; compute $u = \prod_{i=1}^{k} p_i$ and $v = \prod_{i=k+1}^{2k} p_i$; let $\sigma = u \cdot v$; uniformly select two large prime numbers a and b of size $\lambda/2$; find f_1 and f_2 such that $p = f_1 \cdot a \cdot u + 1$ and $q = f_2 \cdot b \cdot v + 1$ are primes; let $m = p \cdot q$ and randomly choose g of order $aubv$ in $\mathbb{Z}_m^*$. The private key is $SK = (p_1, \ldots, p_{2k}, p, q)$, the public key is $PK = (\sigma, g, m)$.

Encrypt$_{PK}(x)$: for $x \in \mathbb{Z}_\sigma$, randomly choose $r \in \mathbb{Z}_m$ and encrypt x as $c = E_{PK}(x) \equiv r^\sigma \cdot g^x \mod m$.

Decrypt$_{SK}(c)$: let $\phi = (p-1)(q-1)$, $c_i \equiv c^{\phi/p_i} \mod m$ and recover, by exhaustive search (p_i is small), $x_i \mod p_i$ such that $x_i = \log_{g^{\phi/p_i}}(c_i) \mod m$. Finally reconstruct x with the Chinese remaindering, $x \equiv CRT(\{x_i, p_i\}) \mod \sigma$.

In the following, cleartexts will be elements of $\mathbb{Z}_\sigma$ while ciphertexts are elements of $\mathbb{Z}_m$. Note that while σ is shared by all players, there is a distinct modulus $\mathbb{Z}_m$ for each player. Consequently a plain text matrix has coefficients in $\mathbb{Z}_\sigma$ but in a layout where each row is encrypted using a different key pk_i, its encryption is no longer a matrix but a sequence of rows over distinct rings $\mathbb{Z}_{m_{pk_i}}$. We will abusively refer to this ciphered data as the ciphered matrix.

Notations. Given some scalar u and a player A, we denote by $\{u\}_A$, as a shortcut to $\{u\}_{pk_A}$, the encryption of data u with the public key of A. This is an element of $\mathbb{Z}_{m_A}$. Similarly, we also denote by $E_A(u)$ the action of encrypting the data u using the public key of A (this means that the player generating this cipher-text knows the plaintext u. For a key sequence K and a matrix A over $\mathbb{Z}_\sigma$, the ciphered matrix obtained by encrypting row i of A by $K[i]$ is denoted by $\{A\}_K$. Row i of $\{A\}_K$ is over $\mathbb{Z}_{m_{K[i]}}$, where $m_{K[i]}$ is the modulus in the public key of player $P_{K[i]}$. We also denote by $r \xleftarrow{\$} D$ the operation of drawing uniformly at random r from a domain D.

2.4 Multiparty protocols security

Here, we recall some widely used notations and results for the security of multiparty protocols.

Definition 3 (from [14]). *Let f be a n-ary functionality, where $f_i(x_1, ..., x_n)$ denotes the i^{th} element of $f(x_1, ..., x_n)$. For $I = \{i_1, ..., i_t\} \subset [n] = \{1, ..., n\}$, we denote by $f_I(x_1, ..., x_m)$ the subsequence $f_{i_1}(x_1, ..., x_n), ..., f_{i_t}(x_1, ..., x_n)$. We let $x_I = (x_{i_1}, ..., x_{i_t})$. Let Π be a n-party protocol for computing f. The view of the i^{th} party during an execution of Π on $\overline{x} = (x_1, ..., x_n)$ is denoted $\mathsf{view}_i^{\Pi}(\overline{x})$, and for I, we let $\mathsf{view}_I^{\Pi}(\overline{x}) = (I, \mathsf{view}_{i_1}^{\Pi}(\overline{x}), ..., \mathsf{view}_{i_t}^{\Pi}(\overline{x}))$. We say that Π securely computes f if there exist a probabilistic polynomial time algorithm, such that for every $I \subset [n]$, we have: $\{S_I((x_I), f_I(\overline{x})), f(\overline{x})\}_{\overline{x}} \overset{C}{\equiv} \{\mathsf{view}_I^{\Pi}(\overline{x}), \mathsf{output}^{\Pi}(\overline{x})\}_{\overline{x}}$.*

Definition 4. *Let $f_1, ..., f_{p(n)}$ be functionalities, and let Π be a protocol. We say that the protocol Π is executed in the $f_1, ..., f_{p(n)}$-hybrid mode if Π uses ideal calls to a trusted party to compute $f_1, ..., f_{p(n)}$.*

Theorem 1 (from [22]). *Let $p(n)$ be a polynomial, let $f_1, ..., f_{p(n)}$ be functionalities, and let $\pi_1, ..., \pi_{p(n)}$ be protocols such that each π_i securely computes f_i in the presence of semi-honest adversaries. Let g be a functionality, and let Π be a protocol that securely computes g in the $f_1, ..., f_{p(n)}$-hybrid model. Then, the protocol $\Pi^{\pi_1, ..., \pi_{p(n)}}$ securely computes g in presence of semi-honest adversaries.*

We will also need a function, which, given a small input is able to securely and deterministically produce a stream of uniformly generated random values. We will achieve this by using classical **mask generation functions**, as defined in [18, Section 10.2]: a function which takes two parameters, a seed Z and a length l and returns a random string of length l. We will then split the output string in as many fragments as needed, and use each of these fragments as a mask. Such function achieve an output indistinguishable property: if the seed is unknown, it is impossible to distinguish between the output of an MGF and a truly random string. Such secure functions exist, see for instance the one given in [18] and in what follows, we will denote by MGF any function that have the aforementioned security properties.

2.5 Relaxing an existing algorithm: `YTP-SS`

The matrix multiplication algorithm using the secure dot-product protocol `YTP-SS` [11, Algorithm 15] is secure against semi-honest adversaries over insecure communication channels. In order to analyze the difference with our proposition, Protocol 7 `MP-SW`, we extract here the core of the former protocol, i.e., without the securization of the channel (that is we remove the protection of the players private elements by random values, and the final communications to derandomize the results). The resulting simplification is called `MP-PDP` and its costs are given in Theorem 2. More details can be found in [11, Algorithm 15]

Theorem 2. *For n players, [11, Algorithm 15], without the channel securization, requires $2(n-1)$ communications. When used to compute a classical matrix product, it requires $n^3 + n(n-1)$ operations overall.*

3 Toolbox

3.1 Initialization Phase

Before the actual computation, the involved parties need to agree on the location and key sequences they will use, generate their key pairs, share their associated public keys, cipher their input data and communicate it where needed. Protocol 1 shows how the input data is initially ciphered and dispatched: each party, identified as $P_i, i \in \{1..N\}$ starts with the i-th row of A and B, and, after generating its own key pair, ciphers its row according to the key sequence.

Protocol 1 `SW-Setup`

Input: Two $N \times N$ matrices A and B over $\mathbb{Z}_\sigma$, where $N = b2^\ell$, such that party P_i knows the i-th row of A and the i-th row B for all $i \in \{1..N\}$. A location and a key sequence $L \in \{1..N\}^N$ and $K \in \{1..N\}^N$ such that (L, K, L, K) form an ℓ-recursive data layout, following Definition 2. All parties know a security parameter λ.

Output: For all $i \in \{1..N\}$, party $P_{L[i]}$ learns vectors $\{a_{i,*}\}_{K[i]}$ and $\{b_{i,*}\}_{K[i]}$ and learns the public key of every other party.

Goal: Generate key pairs for each party, cipher and distribute input matrices according to their respective location and key sequences.

1. **Key generation:** for all $i \in \{1..N\}$, each party P_i locally executes $\texttt{NaccacheSternSetup}(1^\lambda)$ to generate a pair of keys (pk_i, sk_i).
2. **Broadcast keys:** for all $i \in \{1..N\}$, party P_i broadcasts its public key pk_i.
3. **Cipher inputs:** for all $i \in \{1..N\}$, for all $j \in [n]$, party P_i locally performs $\texttt{NaccacheSternEncrypt}(pk_{K[i]}, a_{ij})$ and stores the result as a new vector $\{a_{i,*}\}_{K[i]}$. It does the exact same operation with $b_{i,*}$ to get $\{b_{i,*}\}_{K[i]}$.
4. **Distribute rows:**
   (a) **Rows of A:** for all $i \in \{1..N\}$, party P_i sends $\{a_{i,*}\}_{K[i]}$ to party $P_{L[i]}$.
   (b) **Rows of B:** for all $i \in \{1..N\}$, party P_i sends $\{b_{i,*}\}_{K[i]}$ to party $P_{L[i]}$.

Finally, the protocol sends the ciphered row to the party hosting this row, designated by the location sequence. For input matrices of size N, Protocol 1 requires $2N^2$ communications.

3.2 Multiparty copy

In the various subroutines that compose our algorithm, we will often need to copy and recipher a vector from one Party to another following location and key sequences. For this, one could use proxy reencryption protocols, but it is simpler, in our setting, to instead mask and decrypt, using interaction. Protocol 2 describes protocol `MP-COPY`, performing this very operation for a given ciphered element x hosted by Bob and encrypted for Dan, to its new location at Alice and encrypted for Charlie. Here, Dan is in charge of performing the decryption and the re-encryption of the element. To prevent Dan 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 uses a total of 3 communications.

Protocol 2 `MP-COPY`

Input: Four parties, Alice, Bob, Charlie and Dan. Bob knows a ciphered element $\{x\}_D \in \mathbb{Z}_m$ (for $x \in \mathbb{Z}_\sigma$), ciphered using Dan's public key.

Output: Alice learns the element $\{x\}_C$, ciphered using Charlie's public key.

Goal: Recipher from Dan to Charlie and transfer from Bob to Alice.

1. **Add masking**
   (a) **Random:** Bob samples uniformly at random $r \in \mathbb{Z}_\sigma$
   (b) **Mask:** Bob locally computes $\alpha = \{x\}_D \cdot g^r = \{x + r\}_D \in \mathbb{Z}_m$
   (c) **Communication:** Bob sends α to Dan.
2. **Recipher:**
   (a) **Decipher:** Dan computes $\beta = \texttt{NaccacheSternDecrypt}(sk_D, \alpha) = x + r \in \mathbb{Z}_\sigma$.
   (b) **Cipher:** Dan computes $\gamma = \texttt{NaccacheSternEncrypt}(pk_C, \beta) \in \mathbb{Z}_m$.
   (c) **Communication:** Dan sends γ to Bob.
3. **Remove masking:**
   (a) **Unmask:** Bob locally computes $\delta = \gamma \cdot g^{-r} = \{x\}_C \in \mathbb{Z}_m$
   (b) **Communication:** Bob sends δ to Alice.

3.3 Classical Matrix Multiplication base case

We describe in this section an algorithm to perform classical matrix multiplications in the data and encryption layout of Definition 2. It consists in n^2 scalar products in which, products of elements $a_{i,k}$ of A by elements $b_{k,j}$ of B are performed using the homomorphic multiplication between a ciphertext and a plaintext: $\{a_{i,k}\}_{PK}^{b_{k,j}} = \{a_{i,k}b_{k,j}\}_{PK}$, where PK is the public key that has been used to cipher the element. Therefore, the coefficient $b_{k,j}$ should first be deciphered, and to avoid leaking information, it should also be masked beforehand by some random value.

Protocol 3 takes care of masking and deciphering a whole column of B. There, player Charlie is the only one able to decrypt the masked value $\beta_{k,j} = \{b_{k,j} + t_{k,j}\}_C$. For this we require a stream of uniformly random values $t_{k,j}$, that can be sent. To reduce communications, we here instead use a mask generating function (MGF) that generates this stream from a small seed. Then only the seed need to be communicated to remove the mask. All players have of course to agree beforehand on a choice for this mask generating function.

Protocol 4 shows how player Alice can then recover the ciphertext of one product $\{a_{i,k}b_{k,j}\}_D$. Alice sends her value $\{a_{i,k}\}_D$ to player Charlie who then performs the exponentiation, corresponding to a multiplication on the plaintexts, and sends it back to Alice. Meanwhile Alice has received the seed and generated the masking values $t_{k,j}$ to clean out the product. Finally each coefficient $\{c_{i,j}\}_D$ of the result is computed during a reduction step where player Alice simply multiplies together all corresponding point-wise products.

Protocol 3 `MaskAndDecrypt`

Input: Two parties, further denoted as Bob and Charlie. They both know their own private key, public keys of all the parties involved, the security parameter $\lambda \in \mathbb{N}$ and the modulus $m \in \mathbb{N}$. Moreover, Bob knows a seed $s_k \in \mathbb{N}$ and a ciphered vector of size n, $\{b_{k,*}\}_C$, whose elements $(b_{k,j}) \in \mathbb{Z}_\sigma^n$ have been ciphered using Charlie's public key.

Output: Charlie learns the additively masked plaintext of Bob's input vector.

Goal: Perform the additive masking of Bob's input vector, and let Charlie learn it.

1. **Mask Bob's input:**
   (a) **Generate randoms:** Bob performs MGF(s_k, bitsize(σ)$\times n$) and splits the output in n shares of size bitsize(σ), denoted as $t_{k,j}$ for $j \in \{1..n\}$.
   (b) **Mask vector:** for $j \in \{1..n\}$, Bob computes $\beta_{k,j} = \{b_{k,j}\}_C \cdot g^{t_{k,j}} \in \mathbb{Z}_m$.
   (c) **Communication:** for $j \in \{1..n\}$, Bob sends $\beta_{k,j}$ to Charlie.
2. **Finalize:**
   (a) **Decipher:** for $j \in \{1..n\}$, Charlie performs `NaccacheSternDecrypt`(sk_C, $\beta_{k,j}$) and stores the results in $u_{k,j} = b_{k,j} + t_{k,j} \in \mathbb{Z}_\sigma$.

Protocol 4 `PointwiseProducts`

Input: Four parties, further denoted as Alice, Bob, Charlie and Dan. Alice knows a ciphered $\{a_{i,k}\}_D \in \mathbb{Z}_m$ for given i and k, ciphered using Dan's public key. Bob knows a seed $s_k \in \mathbb{N}$ and Charlie knows a masked vector $(u_{k,*}) \in \mathbb{Z}_\sigma^n$ (each coefficient is masked by a random value).

Output: Alice learns all the ciphertexts $\{a_{i,k}b_{k,j}\}_D$ for $j \in \{1..n\}$.

Goal: Compute the point-wise products for naive matrix product on a given row

1. **Communication:** Alice sends $\{a_{i,k}\}_D$ to Charlie
2. **Multiplication:** for $j \in \{1..n\}$, Charlie computes $\delta_{i,k,j} = \{a_{i,k}\}_D^{u_{k,j}}$, $\delta_{i,k,j} \in \mathbb{Z}_m$
3. **Communication:** for $j \in \{1..n\}$, Charlie sends $\delta_{i,k,j}$ to Alice.
4. **Send seed:** Bob sends s_k to Alice
5. **Generate and remove masks:** Alice performs MGF(s_k, bitsize(σ)$\times n$) and splits the output in n shares of size σ, denoted as $t_{k,j}$ for $j \in \{1..n\}$.
   For $j \in \{1..n\}$, Alice computes:

$$\epsilon_{i,k,j} = \delta_{i,k,j} / \left(\{a_{i,k}\}_D^{t_{k,j}}\right) = \{a_{i,k}(b_{k,j} + t_{k,j}) - a_{i,k}t_{k,j}\}_D \in \mathbb{Z}_m.$$

Overall, Protocol 5 schedules these three operations. In the calls to Protocols `MaskAndDecrypt` and `PointwiseProducts`, Alice is incarnated by Player $P_{L_A[i]}$, Bob by $P_{L_B[k]}$, Charlie by $P_{K_B[k]}$ and Dan by $P_{K_A[i]}$.

Theorem 3. *Protocol 5 correctly computes the product $C = A \times B$ in the specified layout. It requires a communication of $n^3 + 3n^2 + n$ modular integers.*

Proof. Correctness stems first from the fact that $c_{i,j} = \sum_{k=1}^{n} a_{i,k}b_{k,j}$ is obtained "in the exponents" by the homomorphic properties (1). Second the only masks applied, in Protocol 3, are all removed in Protocol 4. Now, the communication cost in number of ring element is n for Protocol 3 and $n + 1$ for Protocol 4. Protocol 3 and Protocol 4 also send one seed, which, for simplicity, we consider smaller than a modular integer. Overall this yields a communication cost lower than $n(n + 1) + n^2(n + 2) = n^3 + 3n^2 + n$ modular integers for Protocol 5. □

Protocol 5 `BaseCase`

Input: two $n \times n$ matrices $\{A\}_{K_A}$ and $\{B\}_{K_B}$ distributed and ciphered according to a base-case data layout $(L_A, K_A, L_B, K_B) \in (\{1..N\}^n)^4$ among parties $(P_1, \ldots, P_N)$ as in Definition 2,

Output: Matrix $C = A \times B$ is distributed and ciphered among parties $(P_1, \ldots, P_N)$ according to the location and key sequences (L_A, K_A).

Goal: Compute $C = A \times B$ distributed and ciphered in the same way as A is.

1. **Computation**:
   For all $k \in \{1..n\}$
   (a) **Choose a seed:** Party $P_{L_B[k]}$ samples uniformly at random a seed $s_k \in \mathbb{N}$ according to the security parameter λ.
   (b) Parties $P_{L_B[k]}$ and $P_{K_B[k]}$ run `MaskAndDecrypt` on vector $\{b_{k,*}\}_{K_B[k]}$
   (c) For all $i \in \{1..n\}$
   Parties $P_{L_A[i]}$, $P_{L_B[k]}$, $P_{K_B[k]}$ and $P_{K_A[i]}$ run `PointwiseProducts` where Parties $P_{L_A[i]}$ learn $\epsilon_{i,k,j} = \{a_{i,k}b_{k,j}\}_{K_A[i]}$ for all $j \in \{1..n\}$.
2. **Reduction**: for all $i \in \{1..n\}$ Party $P_{L_A[i]}$ computes $\{c_{i,j}\}_{K_A[i]} \leftarrow \prod_{k=1}^{n} \epsilon_{i,k,j}$

3.4 Security Analysis

From the formalization of the different protocols we can state the security of the overall base case for matrix multiplication in the following Theorem 4.

Theorem 4. *If players share a* 0*-data-layout, Protocol* `BaseCase` *is secure against one semi-honest adversary.*

The idea is to start by proving the security of the subprotocols and then use the composition theorem and the data layout to prove the security of the double loop of Protocol `BaseCase`. The full formal proof is given in Appendix A.1.

4 Multiparty Strassen-Winograd

4.1 Operation schedule in `MP-SW`

The 22 operations in a recursive step of Strassen-Winograd's algorithm is composed by 15 matrix additions and 7 recursive calls. The matrix additions are performed using component-wise homomorphic additions, denoted by `HOM-MAT-ADD`: each player performs locally a simple homomorphic addition of the rows of the two input operands that she stores. Homomorphic subtraction, denoted by `HOM-MAT-SUB`, works similarly. However, this requires that the two operands share the same key and location sequences. To ensure this, some matrices will be copied from one key-location sequence to another, using a multiparty matrix copy, denoted by `MP-MAT-COPY`. The location sequences of the input an output are non-intersecting (and therefore so are the related key sequences). These operations are achieved by n^2 instances of `MP-COPY` (Protocol 2) as shown in Protocol 6.

Theorem 5. *Assuming a* l*-data layout, Protocol* `MP-MAT-COPY` *is secure against one semi-honest adversary.*

Protocol 6 `MP-MAT-COPY`

Input: an $n \times n$ matrix $\{A\}_{K_A}$ distributed and ciphered according to a location and a key sequence $(L_A, K_A) \in (\{1..N\}^n)^2$ among parties $(P_1, \ldots, P_N)$ following Definition 2 and a location-key sequence (L', K').

Output: A copy $\{A\}_{K'}$ is distributed and ciphered among parties $(P_1, \ldots, P_N)$ according to the location and key sequences (L', K').

For all $i, j \in \{1..n\}^2$
  Parties $P_{L'[i]}, P_{L[i]}, P_{K[i]}$ and $P_{K'[i]}$ run `MP-COPY` to copy $\{a_{i,j}\}_{K[i]}$ to $\{a_{i,j}\}_{K'[i]}$

We only give a sketch of the proof, since its very similar to the one for the `MaskAndDecrypt` protocol within the proof of Theorem 4.

Proof. First, we prove that `MP-COPY` is secure against one semi-honest adversary: from the data layout or the added randomness, each players only see ciphers or additively masked values so that it does not lean anything from the execution. Then, we prove the security in an hybrid model where calls to `MP-COPY` are replaced by an equivalent ideal functionality. Since the output is ciphered accordingly to the data layout, a simulation by ciphering random values is computationally indistinguishable from the real execution. Finally, by sequentially composing calls to the `MP-COPY` protocol, we apply the sequential composition theorem to conclude. □

We propose in Protocol 7 a scheduling of these operations and data movement ensuring that all additions can be made homomorphically, that the key and location sequences for all seven recursive calls satisfy the requirements for a base-case data-layout (Definition 2) and finally that the output matrix also follows the location and key sequences of the first operand. The last three columns in Protocol 7 indicate the location sequences of the input and output operands for each operation.

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 protocol (Protocol 5) requires that these sequences are non-intersecting. In order to satisfy these two constraints the recursive Strassen-Winograd algorithm 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 the base case, Protocol 5.

Lemma 1. *The total communication cost of a recursive level of `MP-SW` following the schedule defined Protocol 7, Step 2 is* $18\left(\frac{n}{2}\right)^2$ *communications.*

Proof. The only communication are that of the 6 calls to `MP-MAT-COPY`, each accounting for $3(n/2)^2$ communication. □

Finally, our main security result is that of the following Theorem 6. The full proof relies on a sequence of hybrid games, where each transition is based on indistinguishability and is given in Appendix A.2.

Protocol 7 `MP-SW`

Input: two $n \times n$ matrices $\{A\}_{K_A}$ and $\{B\}_{K_B}$, distributed and ciphered according to an ℓ-recursive data layout $(L_A, K_A, L_B, K_B) \in (\{1..N\}^n)^4$ among parties $(P_1, \ldots, P_N)$ following Definition 2, where $n = b2^\ell$.

Output: $\{C\}_{K_A} = \{A \times B\}_{K_A}$, distributed and ciphered among parties $(P_1, \ldots, P_N)$ according to the location and key sequences (L_A, K_A).

1. If $\ell = 0$: Parties in (L_A, K_A) and (L_B, K_B) run `BaseCase` on $\{A\}_{K_A}$ and $\{B\}_{K_B}$
2. Else

			In1 loc.	In2 loc.	Out loc.
$\{S_1\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{A_{21}\}_{K_{A_L}}, \{A_{22}\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{A'_{11}\}_{K_{A_L}}$	$\leftarrow$ `MP-MAT-COPY`	$(\{A_{11}\}_{K_{A_U}}, (L_{A_L}, K_{A_L}))$	L_{A_U}		L_{A_L}
$\{S_2\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{S_1\}_{K_{A_L}}, \{A'_{11}\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{S_3\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{A'_{11}\}_{K_{A_L}}, \{A_{21}\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{S'_2\}_{K_{A_U}}$	$\leftarrow$ `MP-MAT-COPY`	$(\{S_2\}_{K_{A_L}}, (L_{A_U}, K_{A_U}))$	L_{A_L}		L_{A_U}
$\{S_4\}_{K_{A_U}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{A_{12}\}_{K_{A_U}}, \{S'_2\}_{K_{A_U}})$	L_{A_U}	L_{A_U}	L_{A_U}
$\{T_1\}_{K_{B_U}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{B_{12}\}_{K_{B_U}}, \{B_{11}\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{B'_{22}\}_{K_{B_U}}$	$\leftarrow$ `MP-MAT-COPY`	$(\{B_{22}\}_{K_{B_L}}, (L_{B_U}, K_{B_U}))$	L_{B_L}		L_{B_U}
$\{T_2\}_{K_{B_U}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{B'_{22}\}_{K_{B_U}}, \{T_1\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{T_3\}_{K_{B_U}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{B'_{22}\}_{K_{B_U}}, \{B_{12}\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{B'_{21}\}_{K_{B_U}}$	$\leftarrow$ `MP-MAT-COPY`	$(\{B_{21}\}_{K_{B_L}}, (L_{B_U}, K_{B_U}))$	L_{B_L}		L_{B_U}
$\{T_4\}_{K_{B_U}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{T_2\}_{K_{B_U}}, \{B'_{21}\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{R_1\}_{K_{A_L}}$	$\leftarrow$ `MP-SW`	$(\{A'_{11}\}_{K_{A_L}}, \{B_{11}\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_2\}_{K_{A_U}}$	$\leftarrow$ `MP-SW`	$(\{A_{12}\}_{K_{A_U}}, \{B_{21}\}_{K_{B_L}})$	L_{A_U}	L_{B_L}	L_{A_U}
$\{R_3\}_{K_{A_U}}$	$\leftarrow$ `MP-SW`	$(\{S_4\}_{K_{A_U}}, \{B_{22}\}_{K_{B_L}})$	L_{A_U}	L_{B_L}	L_{A_U}
$\{R_4\}_{K_{A_L}}$	$\leftarrow$ `MP-SW`	$(\{A_{22}\}_{K_{A_L}}, \{T_4\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_5\}_{K_{A_L}}$	$\leftarrow$ `MP-SW`	$(\{S_1\}_{K_{A_L}}, \{T_1\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_6\}_{K_{A_L}}$	$\leftarrow$ `MP-SW`	$(\{S_2\}_{K_{A_L}}, \{T_2\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_7\}_{K_{A_L}}$	$\leftarrow$ `MP-SW`	$(\{S_3\}_{K_{A_L}}, \{T_3\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R'_1\}_{K_{A_U}}$	$\leftarrow$ `MP-MAT-COPY`	$(\{R_1\}_{K_{A_L}}, (L_{A_U}, K_{A_U}))$	L_{A_L}		L_{A_U}
$\{U_1\}_{K_{A_U}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{R'_1\}_{K_{A_U}}, \{R_2\}_{K_{A_U}})$	L_{A_U}	L_{A_U}	L_{A_U}
$\{U_2\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{R_1\}_{K_{A_L}}, \{R_6\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U_3\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{U_2\}_{K_{A_L}}, \{R_7\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U_4\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{U_2\}_{K_{A_L}}, \{R_5\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U'_4\}_{K_{A_U}}$	$\leftarrow$ `MP-MAT-COPY`	$(\{U_4\}_{K_{A_L}}, (L_{A_U}, K_{A_U}))$	L_{A_L}		L_{A_U}
$\{U_5\}_{K_{A_U}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{U'_4\}_{K_{A_U}}, \{R_3\}_{K_{A_U}})$	L_{A_U}	L_{A_U}	L_{A_U}
$\{U_6\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-SUB`	$(\{U_3\}_{K_{A_L}}, \{R_4\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U_7\}_{K_{A_L}}$	$\leftarrow$ `HOM-MAT-ADD`	$(\{U_3\}_{K_{A_L}}, \{R_5\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}

3. **End result** $\{C\}_{K_A} \leftarrow \begin{bmatrix} \{U_1\}_{K_{A_U}} & \{U_5\}_{K_{A_U}} \\ \{U_6\}_{K_{A_L}} & \{U_7\}_{K_{A_L}} \end{bmatrix}$

Theorem 6. *Assuming an ℓ-data layout, Protocol `MP-SW` is secure against one semi-honest adversary.*

4.2 Finalization step

Finally, there remains to decipher and distribute each row of $\{C\}_{K_A}$ to the party who has to learn it. By setting the key sequence to $K_A = (1, 2, 3 \dots)$ as in Equation (1), this player is able to perform the decryption himself. This finalization step is formally described in Protocol 8 and uses N^2 communications.

Protocol 8 `SW-Finalize`

Input: An $N \times N$ matrix $\{C\}_{K_C}$ distributed and ciphered according to the location and key sequences $(L_C, K_C) \in (\{1..N\}^N)^2$ among parties $P_1, \dots, P_N$, following Definition 1.

Output: Each party $P_{K_C[i]}$ learns the plaintext of the i-th row of C.

The protocol

1. **Exchange rows:** For all $i \in \{1..N\}$, party $P_{L_C[i]}$ send row i of C to party $P_{K_C[i]}$.
2. **Decipher vector:** For all $i \in \{1..N\}$, for all $j \in \{1..N\}$, party $P_{K_C[i]}$ runs $\texttt{NaccacheSternDecrypt}(sk_{K_C[i]}, (\{c_{i,j}\}_{K_C[i]}))$ and stores the output values in a vector $c_{K_C[i]} \in \mathbb{Z}_\sigma^N$.

4.3 Cost and security analysis

From Lemma 1 and Theorem 3, the recurrence relation for communication complexity of `MP-SW` writes:

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

The threshold at which the recursive algorithm should switch to the base case algorithm is set by finding at which dimension b does the base case algorithm start to perform worse than one recursive level. In terms of communication cost, this means the following equation: $7((\frac{b}{2})^3 + 3(\frac{b}{2})^2 + 3(\frac{b}{2})) + 18(\frac{b}{2})^2 = b^3 + 3b^2 + b$ which comes from injecting the base case cost of Theorem 3 into the recurrence formula. It gives a threshold of $b = 56$.

Theorem 7. *For $N = b2^\ell$ parties $(P_i)_{i \in \{1..N\}}$ and two matrices $A, B \in \mathbb{Z}_\sigma^{N \times N}$, such that party P_i knows the i-th row of A and the i-th row B for all $i \in \{1..N\}$, the execution in sequence of algorithms (`SW-Setup`; `MP-SW`; `SW-Finalize`), using the ℓ-recursive data layout of Equation (1), correctly computes $C = A \times B \in \mathbb{Z}_\sigma$ with $O(7^\ell b^3)$ communications in $O(\ell)$ rounds and is secure against one semi-honest adversary. When b is constant, then $\ell = O(\log_2(N))$, and the communication bound is $O(N^{\log_2(7)})$.*

Proof. Correctness of `MP-SW` is given by Theorem 3 for the basecase and that of Strassen-Winograd algorithm (Section 2.1). Then `SW-Setup` is just the set up of the keys and initial encipherings, while `SW-Finalize` is the associated decipherings. Then, the communication bound stems from Theorem 3 and Equation (2),

with $3N^2$ communications for `SW-Setup` and `SW-Finalize`. The non-recursive parts of each recursive level of `MP-SW` require a constant number of rounds, and so does the execution of the `BaseCase`, leading to a total of $O(\ell)$ rounds. For the security. again `SW-Setup` is just the communication of public keys and self-ciphered values, while `SW-Finalize` is also the communication of ciphered values to their legitimate locations. Finally, Theorem 6 asserts the security of `MP-SW` and the sequential execution of (`SW-Setup`; `MP-SW`; `SW-Finalize`) that of the whole process. □

We now compare the cost of `MP-SW` with the cost of `MP-PDP`, $C_{\texttt{MP-PDP}}(n) = n^3+n(n-1)$. We also recall that the initialization step `SW-Setup` costs $C_{\text{init}} = 2n^2$ and the finalization step `SW-Finalize` costs $C_{\text{final}} = n^2$. The crossover point where our full algorithm improves over `MP-PDP` in communication cost is obtained by solving the equation: $C(n) + 3n^2 \leq n^3 + n(n-1)$ which yields $n > 94$, with one recursive call. This means that for any instance of dimension larger than 96, the proposed `MP-SW` algorithm has a better communication cost than `MP-PDP`.

5 Experiments

We implemented the algorithms under study[6] to demonstrate their behavior in practice and compared them to the state of the art implementations of other solutions. In the following `SPD`$\mathbb{Z}_{2^k}$ refers to a run of a textbook matrix multiplication algorithm performed with the general purpose library `SPD`$\mathbb{Z}_{2^k}$ [4][7], `YTP-SS` refers to n^2 applications of [11, Algorithm 15]; `MP-PDP` refers the relaxation and improvement of this algorithm to the current setting; `MP-SW` refers to our implementation of Protocol 7 using Protocol 5 as a basecase with threshold set to $n = 56$. 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).

Please note that, while `MP-PDP` and `MP-SW` share the same security model, `YTP-SS` and `SPD`$\mathbb{Z}_{2^k}$ achieve better security: malicious adversaries over insecure channels. Also, `SPD`$\mathbb{Z}_{2^k}$ uses a different approach based on oblivious transfer and secret sharing. However, as they were the only state of the art implementations available, we still chose to include them in our comparisons.

Figure 2 presents the volume of communication performed by these four variants. Note that the cross-over point of $n = 96$ between `MP-PDP` and `MP-SW` is confirmed experimentally. For `SPD`$\mathbb{Z}_{2^k}$, computations were performed for small matrices only because of computational power requirements: on a workstation with 16 GB of RAM and an Intel i5-7300U @2.60GHz, computations stalled for any matrices larger than 37×37.

To reach this communication improvement, the price to pay is that of some computational slowdown, as shown in Table 1

[6] `C++` source files, including benchmarks for `YTP-SS` and `SPD`$\mathbb{Z}_{2^k}$, are available on request via the PC chair and will be made publicly available if the paper is accepted.
[7] https://github.com/bristolcrypto/SPDZ-2

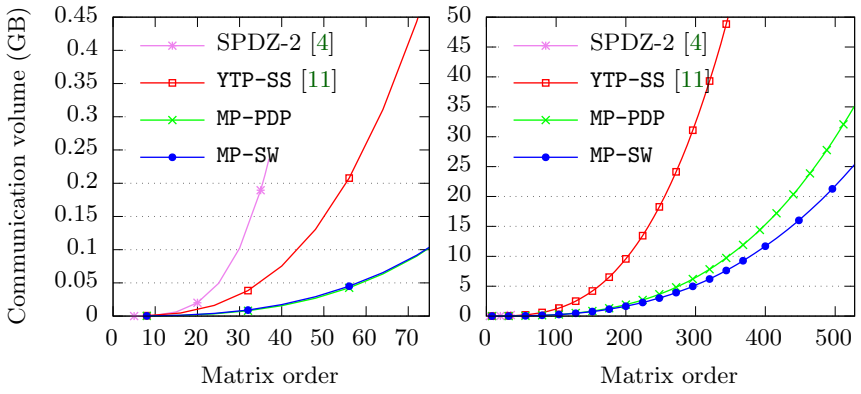

Fig. 2. Comparing communication volume for multiparty matrix multiplications.

Table 1. Computation time (in s) per player of Multiparty Strassen-Winograd `MP-SW` compared to `MP-PDP` on an Intel Xeon E7-8860 2.2Ghz.

Key size	1024			2048		
n	16	32	64	16	32	64
`MP-PDP`	0.58	2.68	11.01	4.54	18.05	69.80
`MP-SW`	2.87	6.19	13.27	23.63	49.22	196.24

However, with the same order of magnitude for the computational cost and the communication cost, communications should be largely dominant. Therefore, the improvement in communication volume (for instance about 27.8% for $n = 528$ players between algorithms `MP-SW` and `MP-PDP`) is the one that matters.

6 Conclusion and Perspective

We have presented in this paper a novel secure multiparty matrix multiplication where each player owns one row of the different matrices. For this we use Strassen-Winograd algorithm and reduce for the first time the total communication volume from $O(N^3)$ to $O(N^{\log_2(7)})$. The improvement in communication cost over state of the art algorithms takes effect for dimension as small as 96.

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 improvements could be envisioned. For instance, removing the need for players to encrypt their share of the B matrix beforehand. While this is required in order to preserve security, a large part of the computing cost lies in the operations required to decipher and re-cipher that data. In particular, since the `MP-COPY` protocol nothing more than a straight-

forward proxy re-encryption, and we want to further investigate the benefits of dedicated proxy re-encryption techniques available in this context. Another possibility would be to replace the Naccache-Stern by a faster cryptosystem. The difficulty is to be able to combine the masking schemes with the homomorphic encryption.

A Security proofs

A.1 Base case security proof

Theorem 4 (From Section 3.4). *If players shares a* 0*-data-layout, Protocol* `BaseCase` *is secure against one semi-honest adversary.*

Proof. We start by proving that both subprotocols `MaskAndDecrypt` and `PointwiseProducts` are secure against one semi-honest adversary.

Protocol `MaskAndDecrypt` is a 2-party protocol such that:
$\mathsf{output}^{\texttt{M\&D}}((\{\boldsymbol{b_1}\}_{P_2}, s_1), -) = (-, \boldsymbol{u_1})$, with $\boldsymbol{b_1} = b_{k,*}$, $s_1 = s_k$, $u_1 = \{u_{k,j}\}_{j\in\{1..n\}}$. The proof is then divided in two parts: one for each corruption case. We labeled P_1 the player providing the seeds as input.

P_1 is corrupted: The view of P_1 is: $\mathsf{view}_{P_1}^{M\&D} = (\boldsymbol{t_1}, \boldsymbol{\beta_1})$ From the inputs of P_1, the simulator is able to perfectly simulate the view of P_1.

P_2 is corrupted: The view of P_2 is: $\mathsf{view}_{P_2}^{M\&D} = (\boldsymbol{\beta_1})$ From the output $\boldsymbol{u_1}$ of P_2, the simulator S_2 ciphers each of its elements with the key of P_2. From the IND-CPA security, the simulated view is computationally indistinguishable from the real one.

Protocol `PointwiseProducts` is a 4-parties protocol. However, the 4^{th} player does not have any input nor output: only its public key is used. In the same vein, P_2 only sends s_2 and does not interact otherwise. Its view is empty, so that its simulator is trivial. Therefore, the proof is only divided in two part. The output of the protocol is: $\mathsf{output}^{\texttt{PWP}}(\{a_1\}_{P_4}, s_2, \boldsymbol{u_k}, -) = (\boldsymbol{\epsilon}, -, -, -)$ with $a_1 = a_{i,k}$, $\boldsymbol{u_k} = u_{k,*}$ and $\boldsymbol{\epsilon} = \{\epsilon_{i,k,j}\}_{j\in\{1..n\}}$.

P_1 is corrupted: The view of P_1 is: $\mathsf{view}_{P_1}^{PWP} = (s_2, \boldsymbol{t_1}, \boldsymbol{\delta_1})$ The simulator S_1 is the following: picks $s_2' \xleftarrow{\$} \mathbb{Z}_\sigma$ computes $\boldsymbol{t_1'}$ as in the protocol. Then, from the output $\boldsymbol{\epsilon}$ and the input $\{a_1\}_{P_4}$, it computes $\boldsymbol{\delta'} = \boldsymbol{\epsilon} * \{a_1\}_{P_4}^{\boldsymbol{t_1'}}$ component wise. Since the $\boldsymbol{\delta}$ values are ciphered with the key of P_4, and that s_1 is a random value, both views are indistinguishable.

P_3 is corrupted: We have $\mathsf{view}_{P_3}^{PWP} = (\{a_1\}_{P_4}, \boldsymbol{\delta_1})$ S_3: $a_1' \xleftarrow{\$} \mathbb{Z}_\sigma$, then the value is ciphered with the public key of P_4 to obtain $\{a_1'\}_{P_4}$. Next, it computes $\boldsymbol{\delta_1'}$ as in protocol using the simulated value $\{a_1'\}_{P_4}$. This simulation is computationally indistinguishable from the real view thanks to the IND-CPA security of the cryptosystem.

We denote by $F^{M\&D}$ (respectively F^{PWP}) the ideal functionalities associated to the protocol MaskAndDecrypt (resp. PointWiseProduct). If players shares a 0-data-layout, the BaseCase protocol is secure against one semi-honest adversary in the $(F^{M\&D}, F^{PWP})$-hybrid model.

BaseCase is a N-party protocol, where the view depends on which group the player belongs. Since players share a 0-data-layout, there are four distinct possibilities: $\{L_A, K_A, L_B, K_B\}$. The cases where $P_{K_A[i]}$ or $P_{L_B[i]}$ is corrupted are trivial, since their respective view are empty in the $(F^{M\&D}, F^{PWP})$-hybrid model.

$P_{L_A[i]}$ is corrupted: The view of $P_{L_A[i]}$ is: $\mathsf{view}_{P_{L_A[i]}}^{BC} = (\{\boldsymbol{\epsilon}\}P_{K_A[i]})$ where ϵ_i is the output of a call to F^{PWP}. The simulator S_i executes: for each $k \in \{1..N\}$: from the output of the ideal functionality F^{BC}, it picks $N-1$ random shares in $\mathbb{Z}_\sigma$ the (denoted $\epsilon_i', i \in \{1..N-1\}$), and ciphers them using $Pk_{K_A}[i]$. Then, it chooses the last share ϵ_n' such that: $\epsilon_n' * \prod_{k=1}^{n-1} \epsilon_i'$. If ϵ_n' belongs to $\mathbb{Z}_m$, then it outputs each component of $\boldsymbol{\epsilon'}$, otherwise it redoes the process from the beginning for the k^{th} step. The definition of the data layout assures that $P_{L_A[i]} \neq P_{K_A[i]}$, so that ϵ_i and ϵ_i' are indistinguishable as long as the encryption scheme is IND-CPA. Moreover, since the choice of each share is

consistent with the output of the protocol (i.e., their product is equal to the output), the adversary is not able to computationally distinguish between the real and the simulated execution.

$P_{L_B[i]}$ is corrupted: The view of $P_{L_B[i]}$ is: $\{\mathsf{view}^{BC}_{P_{L_B[i]}} = (\boldsymbol{u})\}$. The output of the protocol is empty for this player. The simulator picks n random values from $\mathbb{Z}_\sigma$, and outputs each of them to form $\boldsymbol{u'}$. In the real world, each u_i is masked by a random value (unknown by $P_{L_B[i]}$ since $P_{L_B[i]} \neq P_{K_B[i]}$), so that u_i and u'_i are then perfectly indistinguishable. □

Finally, we apply the composition Theorem 1: since we have proven the security of Π^{BC} in the $(F^{M\&D}, F^{PWP})$-hybrid model, and that the protocols $\Pi^{M\&D}$ and Π^{PWP} are secure, and that each call to the both of these protocols are sequentially made, we conclude that the BaseCase protocol is secure against one semi-honest adversary. Moreover, the 0-data layout ensures that the seed sharing does not leak information.

A.2 Multiparty Strassen-Winograd security proof

Theorem 6 (From Section 4.1). *Assuming an ℓ-data layout, Protocol MP-SW is secure against one semi-honest adversary.*

Proof. First, we prove that MP-SW is secure is the $F^{BaseCase}, F^{Copy}, F^{\texttt{MP-SW}}_{N/2}$-hybrid model, where $F^{BaseCase}$, F^{Copy} and $F^{\texttt{MP-SW}}_{N/2}$ respectively denotes the ideal functionality associated to the protocol BaseCase, MP-MAT-COPY, and MP-SW with $N/2$ players. In this model, calls to MP-MAT-COPY are replaced by ideals calls to F^{Copy}. In the same vein, if $N \leq T$, the MP-SW calls are replaced by $F^{BaseCase}$, or by $F^{\texttt{MP-SW}}_{N/2}$ otherwise. We need to prove that for any corrupted player, its real view is indistinguishable from the simulated one. In the following, the inverse notation associated to the data layout returns the index of associated to the value $L[i]$ or $K[i]$.

From the inputs are as described in MP-SW (implicit in the following), the outputs for the player $P_{L_{A_X[i]}}$ are the rows of the following matrices, ciphered with $Pk_{K_{A_X[i]}}$, with $X \in \{U, L\}$. $\mathsf{output}^{\texttt{MP-SW}}_{A_U}(U_1, U_5)$ and $\mathsf{output}^{MP-SW}_{A_L}(U_6, U_7)$. Using the same notations, we obtains the following views:
$\mathsf{view}^{\texttt{MP-SW}}_{L_{A_U}} = (S'_2, S_4, R'_1, R_2, R_3, U'_4)$,
$\mathsf{view}^{\texttt{MP-SW}}_{L_{A_L}} = (S_1, A'_{11}, S_2, S_3, R_1, R_4, R_5, R_6, R_7, U_2, U_3, U_4)$,
$\mathsf{view}^{\texttt{MP-SW}}_{L_{B_U}} = (T_1, B'_{22}, T_2, T_3, T_4)$, $\mathsf{view}^{\texttt{MP-SW}}_{L_{B_L}} = (-)$

We construct a generic simulator, where differences depending on the corrupted player are explicitly detailed. The simulator $S_{i\in\{1..N\}}$ takes two random matrices in α and β both in $\mathbb{Z}_\sigma^{(N*N)}$. Then, it replaces the rows for the corrupted player with its actual inputs (i.e., the rows of A and B owned by the corrupted player). The remaining coefficients are ciphered accordingly to the data layout. The first part of the protocol (i.e. the computation of S_i and T_i, $i \in \{1..4\}$) is simulated using the inputs and ideal calls to F^{Copy}. This simulates the views for the L_B cases. Then, there are two cases.

$P_{L_{A_U[i]}}$ is corrupted: From the output, the simulator $S_{L_{A_U[i]}}$ takes $N/2$ random values from $\mathbb{Z}_m$ to obtain the simulation of the row of U_4. Then, it computes the row of $R_3 = \texttt{HOM-MAT-SUB}(U_5, U_4)$. Similarly, it take the row R'_1 at random, and computes $R_1 = \texttt{HOM-MAT-SUB}(U_1, R'_1)$.

$P_{L_{A_L[i]}}$ is corrupted: $S_{L_{A_L[i]}}$ samples $3N/2$ random values from $\mathbb{Z}_m$ to simulate the row of U_2, R_1 and R_7. Next, it computes: $R_6 = \texttt{HOM-MAT-SUB}(U_2, R_1)$, $U_3 =$

$\texttt{HOM-MAT-ADD}(U_2, R_7)$, $R_4 = \texttt{HOM-MAT-SUB}(U_3, U_6)$, $R_5 = \texttt{HOM-MAT-SUB}(U_7, U_3$ and $U_4 = \texttt{HOM-MAT-ADD}(U_2, R_5)$.

We now prove that the simulated view is indistinguishable from the real one. The proof relies on a sequence of hybrid games, where each transition is based on indistinguishability.

H_0: The first game represents the view of a real protocol execution. in the $F^{\texttt{MP-SW}}, F^{Copy}, F^{\texttt{MP-SW}}_{N/2}$-hybrid model.

H_1: for each call to F^{Copy}, we replace the output of the functionality by random numbers, accordingly ciphered with the data layout. i.e.: $\forall j \in \{1..N\}, r_i \xleftarrow{\$} \mathbb{Z}_\sigma$ and $\{r_j\}_{K_X[j]}$ with $X \in \{A_L, B_U\}$. Since only one player is corrupted, and (L_A, K_A, L_B, K_B) is a l-recursive data layout which verifies $(L_{X_U} \cup K_{X_U}) \cap (L_{X_L} \cup K_{X_L}) = \emptyset, X \in \{A, B\}$, then the player obtains ciphers which it cannot decipher. Then, the IND-CPA security of the cryptosystem ensures that H_0 and H_1 are indistinguishable.

H_2: In this game, we replace the output obtained from: F^{BC}) if $N \leq T$; $F^{\texttt{MP-SW}}$ with $N/2$ players otherwise; by the previously detailed simulation for the R_i, $i \in \{1..N\}$. From the data layout, the corrupted player ($P_{L_{A_U}[i]}$ or $P_{L_{A_L}[i]}$) in the real case gets undecipherable values that cannot be guessed from the inputs of the adversary (which knows one row of each matrix in the worst case) so that the simulation is computationally indistinguishable. Then, $H_1 \overset{C}{\equiv} H_2'$.

H_3: In this game, we replace the U_i of the real view with the simulated ones U_i, $i \in \{2..4\}$. Each of the simulated values is directly computed from the output, so that as long as the adversary is not able to distinguish ciphers, the simulation is computationally indistinguishable from the real execution.

H_3 represents the simulated view for N players. We have then proven that $\texttt{MP-SW}$ is secure against one semi-honest adversary in the F^{Copy}, F^{BC}-model.

Second, we prove that if we assume a l-data layout between the players, the $\texttt{MP-SW}$ protocol is secure against one semi-honest adversary under a sequential composition of the sub-protocols MP-MAT-COPY and BaseCase. By induction, we suppose that the protocol $\texttt{MP-SW}$ is secure with $N/2$ players, and we show that the protocol $\texttt{MP-SW}$ for N players.

Base Case : $N \leq T$. In this case, $\texttt{MP-SW}$ is replaced by the BaseCase protocol. By construction, the data layout is now 0-recursive. Then, the corrupted player cannot act as more than one player in the execution, so that the security of the protocol against one-semi honest is enough.

Induction : $N > T$. In this case, each call to the $\texttt{MP-SW}$ protocol is assumed secure from the induction hypothesis. Then, each of these calls can be realized sequentially.

Then, since all sub-protocols calls can be realized sequentially, and since we have proven that $\texttt{MP-SW}$ is secure in the $F^{\texttt{MP-SW}}, F^{Copy}, F^{\texttt{MP-SW}}_{N/2}$-hybrid model, the sequential composition theorem ensures that the protocol obtained by composition is also secure. Henceforth, by induction, we have proven that from $F^{\texttt{MP-SW}}_{N/2}$, we are able to construct a secure execution of $\texttt{MP-SW}$. In conclusion, the protocol $\texttt{MP-SW}$ is secure against one semi-honest adversary. □

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. 1974.
2. B. Boyer and J.-G. Dumas. Matrix multiplication over word-size modular rings using approximate formulas. *ACM Trans. Math. Softw.*, 2016.

3. R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. 2015.
4. R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPDZ_{2^k}: Efficient mpc mod 2^k for dishonest majority. In *Advances in Cryptology - CRYPTO 2018*.
5. Ö. Dagdelen and D. Venturi. A multiparty protocol for privacy-preserving cooperative linear systems of equations. In *BalkanCryptSec 2014*.
6. I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO 2017*.
7. D. Demmler, T. Schneider, and M. Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*.
8. W. Du and Z. Zhan. A practical approach to solve secure multiparty computation problems. In *NSPW'02*, 2002.
9. J.-G. Dumas, P. Giorgi, and C. Pernet. Dense linear algebra over word-size prime fields: The FFLAS and FFPACK packages. *ACM Trans. Math. Softw.*, 2008.
10. 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*, 2013.
11. J.-G. Dumas, P. Lafourcade, J.-B. Orfila, and M. Puys. Dual protocols for private multiparty matrix multiplication and trust computations. *Computers & Security*, 2017.
12. A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017.
13. 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, *ICISC 2004*.
14. O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. 2004.
15. Y. Ishai, M. Mittal, and R. Ostrovsky. On the message complexity of secure multiparty computation. In *PKC 2018*.
16. S. Jarecki. Efficient covert two-party computation. In *PKC 2018*.
17. A. Josang. Probabilistic logic under uncertainty. In *13th Computing: Australasian Theory Symposium (CATS2007)*, 2007.
18. B. Kaliski and J. Staddon. RSA Cryptography Specifications (PKCS #1). RFC 2437, 1998.
19. I. Kaporin. A practical algorithm for faster matrix multiplication. In *Numerical Linear Algebra with Applications*, 1999.
20. E. Karstadt and O. Schwartz. Matrix multiplication, a little faster. SPAA '17.
21. F. Le Gall. Powers of tensors and fast matrix multiplication. ISSAC '14.
22. Y. Lindell. How to simulate it – a tutorial on the simulation proof technique, 2017.
23. P. K. Mishra, D. Rathee, D. H. Duong, and M. Yasuda. Fast secure matrix multiplications over ring-based homomorphic encryption. Cryptology ePrint Archive, Report 2018/663, 2018.
24. D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. CCS '98, 1998.
25. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, 1999.
26. P. Rindal and M. Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *USENIX Security Symposium, 2016*.
27. A. Shamir. How to share a secret. *Comm. ACM*, 1979.
28. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 1969.