



**HAL**  
open science

# Private Multi-party Matrix Multiplication and Trust Computations

Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, Maxime Puys

► **To cite this version:**

Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, Maxime Puys. Private Multi-party Matrix Multiplication and Trust Computations. 13th International Conference on Security and Cryptography SECRYPT 16, Jul 2016, Lisbon, Portugal. pp.61-72, 10.5220/0005957200610072 . hal-01344750v2

**HAL Id: hal-01344750**

**<https://hal.univ-grenoble-alpes.fr/hal-01344750v2>**

Submitted on 14 Sep 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Private Multi-party Matrix Multiplication and Trust Computations\*

Jean-Guillaume Dumas<sup>1</sup>, Pascal Lafourcade<sup>2</sup>, Jean-Baptiste Orfila<sup>1</sup> and Maxime Puys<sup>3</sup>

<sup>1</sup> Université Grenoble Alpes, CNRS, LJK, 700 av. centrale, IMAG - CS 40700, 38058 Grenoble cedex 9, France

<sup>2</sup> University Clermont Auvergne, LIMOS, Campus Universitaire des Cézéaux, BP 86, 63172 Aubière Cedex, France

<sup>3</sup> Université Grenoble Alpes, CNRS, Verimag, 700 av. centrale, IMAG - CS 40700, 38058 Grenoble cedex 9, France  
{Jean-Guillaume.Dumas, Jean-Baptiste.Orfila, Maxime.Puys}@imag.fr, Pascal.Lafourcade@udamail.fr

Keywords: Secure Multiparty Computation (MPC); Distributed Matrix Multiplication; Trust evaluation; Proverif.

Abstract: This paper deals with distributed matrix multiplication. Each player owns only one row of both matrices and wishes to learn about one distinct row of the product matrix, without revealing its input to the other players. We first improve on a weighted average protocol, in order to securely compute a dot-product with a quadratic volume of communications and linear number of rounds. We also propose a protocol with five communication rounds, using a Paillier-like underlying homomorphic public key cryptosystem, which is secure in the semi-honest model or secure with high probability in the malicious adversary model. Using ProVerif, a cryptographic protocol verification tool, we are able to check the security of the protocol and provide a countermeasure for each attack found by the tool. We also give a randomization method to avoid collusion attacks. As an application, we show that this protocol enables a distributed and secure evaluation of trust relationships in a network, for a large class of trust evaluation schemes.

## 1 Introduction

Secure multiparty computations (MPC), introduced by Yao (Yao, 1982) with the millionaires' problem, has been intensively studied during the past thirty years. The idea of MPC is to allow  $n$  players to jointly compute a function  $f$  using their private inputs without revealing them. In the end, they only know the result of the computation and no more information. Depending on possible corruptions of players, one may prove that a protocol may resist against a collusion of many players, or that it is secure even if attackers try to maliciously modify their inputs. Mostly any function can be securely computed (Ben-Or et al., 1988) and many tools exist to realize MPC protocols. They comprise for instance the use of a Trusted Third Party (Du and Zhan, 2002), the use of Shamir's secret sharing scheme (Shamir, 1979), or more recently the use of homomorphic encryption (Goethals et al., 2005). It is also possible to mix these techniques (Damgård et al., 2012).

Our goal is to apply MPC to the a distributed eval-

uation of trust, as defined in (Jøsang, 2007; Dumas and Hossayni, 2012). There, confidence is a combination of degrees of trust, distrust and uncertainty between players. Aggregation of trusts between players on a network is done by a matrix product defined on two monoids (one for the addition of trust, the other one for multiplication, or transitivity): each player knows one row of the matrix, its partial trust on its neighbors, and the network as a whole has to compute a distributed matrix squaring. Considering that the trust of each player for his colleagues is private, at the end of the computation, nothing but one row of the global trust has to be learned by each player (*i.e.*, nothing about private inputs should be revealed to others). Thus, an MPC protocol to resolve this problem should combine privacy (nothing is learned but the output), safety (computation of the function does not reveal anything about inputs) and efficiency (Lindell, 2009). First, we need to define a MPC protocol which allows us to efficiently compute a distributed matrix product with this division of data between players. The problem is reduced to the computation of a dot product between vectors  $U$  and  $V$  such that one player knows  $U$  and  $V$  is divided between all players.

**Related Work.** Dot product in the MPC model has been widely studied (Du and Atallah, 2001; Amirbekyan and Estivill-Castro, 2007; Wang et al., 2008).

---

\*This work was partially supported by "Digital trust" Chair from the University of Auvergne Foundation, by the HPAC project (ANR 11 BS02 013), the ARAMIS project (PIA P3342-146798) and the LabEx PERSYVAL-Lab (ANR-11-LABX-0025).

However, in these papers, assumptions made on data partitions are different: there, each player owns a complete vector, and the dot product is computed between two players where; in our setting, trust evaluation should be done among peers, like certification authorities. For instance, using a trusted third party or permuting the coefficients is unrealistic. Now, computing a dot product with  $n$  players is actually close to the MPWP protocol of (Dolev et al., 2010), computing a mean in a distributed manner: computing dot products is actually similar to computing a weighted average where the weights are in the known row, and the values to be averaged are privately distributed. In MPWP the total volume of communication for a dot product is  $O(n^3)$  with  $O(n)$  communication rounds. Other generic MPC protocols exist, also evaluating circuits, they however also require  $O(n^3)$  computations and/or communications per dot-product (Bendlin et al., 2011; Damgård et al., 2012).

**Contributions.** We provide the following results:

- A protocol *P-MPWP*, improving on *MPWP*, which reduces both the computational cost, by allowing the use of Paillier’s cryptosystem, and the communication cost, from  $O(n^3)$  to  $O(n^2)$ .
- An  $O(n)$  time and communications protocol *Distributed and Secure Dot-Product (DSDP<sub>i</sub>)* (for  $i$  participants) which allows us to securely compute a dot product  $UV$ , against a semi-honest adversary, where one player owns a vector  $U$  and where each player knows one coefficient of  $V$ .
- A parallel variant that performs the dot-product computation in parallel among the players, limits the total number of rounds. This is extended to a *Parallel Distributed and Secure Matrix-Multiplication (PDSMM<sub>i</sub>)* family of protocols.
- A security analysis of the *DSDP* protocol using a cryptographic protocol verification tool, here ProVerif (Blanchet, 2001; Blanchet, 2004). This tool allows us to define countermeasures for each found attack: adapted proofs of knowledge in order to preserve privacy and a *random ring order*, where private inputs are protected as in a wiretap code (Ozarow and Wyner, 1984) and where the players take random order in the protocol to preserve privacy with high probability, even against a coalition of malicious insiders.
- Finally, we show how to use these protocols for the computation of trust aggregation, where classic addition and multiplication are replaced by more generic operations, defined on monoids.

In Section 2, we thus first recall some multi-party computation notions. We then introduce in Section 3 the trust model based on monoids. In Section 4, we present our quadratic variant of MPWP and a linear-

time protocol in Section 5. We then give the associated security proofs and countermeasures in Section 6 and present parallelized version in Section 7. Finally, in Section 8, we show how our protocols can be adapted to perform a private multi-party trust computation in a network.

## 2 Background and Definitions

We use a public-key homomorphic encryption scheme where both addition and multiplication are considered. There exist many homomorphic cryptosystems, see for instance (Mohassel, 2011, § 3) and references therein. We need the following properties on the encryption function  $E$  (according to the context, we use  $E_{PubB}$ , or  $E_1$  or just  $E$  to denote the encryption function, similarly for the signature function,  $D_1$  or  $D_{PrivB}$ ): computing several modular additions, denoted by  $Add(c_1; c_2)$ , on ciphered messages and one modular multiplication, denoted by  $Mul(c; m)$ , between a ciphered message and a cleartext. That is,  $\forall m_1, m_2 \in \mathbb{Z}/m\mathbb{Z}$ :  $Add(E(m_1); E(m_2)) = E(m_1 + m_2 \bmod m)$  and  $Mul(E(m_1); m_2) = E(m_1 m_2 \bmod m)$ . For instance, Paillier’s or Benaloh’s cryptosystems (Paillier, 1999; Benaloh, 1994; Fousse et al., 2011) can satisfy these requirements, *via* multiplication in the ground ring for addition of enciphered messages ( $Add(E(m_1); E(m_2)) = E(m_1)E(m_2) \bmod m$ ), and *via* exponentiation for ciphered multiplication ( $Mul(E(m_1); m_2) = E(m_1)^{m_2} \bmod m$ ), we obtain the following homomorphic properties:

$$E(m_1)E(m_2) = E(m_1 + m_2 \bmod m) \quad (1)$$

$$E(m_1)^{m_2} = E(m_1 m_2 \bmod m) \quad (2)$$

Since we consider the semantic security of the cryptosystem, we assume that adversaries are probabilistic polynomial time machines. In MPC, most represented intruders are the following ones:

- *Semi-honest (honest-but-curious) adversaries*: a corrupted player follows the protocol specifications, but also tries to gather as many information as possible in order to deduce some private inputs.
- *Malicious adversaries*: a corrupted player that controls the network and stops, forges or listens to messages in order to gain information.

## 3 Monoids of Trust

There are several schemes for evaluating the transitive trust in a network. Some use a single value representing the probability that the expected action will

happen; the complementary probability being an uncertainty on the trust. Others include the *distrust* degree indicating the probability that the opposite of the expected action will happen (Guha et al., 2004). More complete schemes can be introduced to evaluate trust: Jøsang introduces the *Subjective Logic* notion which expresses beliefs about the truth of propositions with degrees of "uncertainty" in (Jøsang, 2007). Then the authors of (Huang and Nicol, 2010) applied the associated calculus of trust to public key infrastructures. There, trust is represented by a triplet, (trust, distrust, uncertainty) for the proportion of experiences proved, or believed, positive; the proportion of experiences *proved negative*; and the proportion of experiences with unknown character. As *uncertainty* =  $1 - \text{trust} - \text{distrust}$ , it is sufficient to express trust with two values as  $\langle \text{trust}, \text{distrust} \rangle$ . In e.g. (Foley et al., 2010) algorithms are proposed to quantify the trust relationship between two entities in a network, using transitivity and reachability. For instance, in (Dumas and Hossayni, 2012) the authors use an adapted power of the adjacency matrix to evaluate the trust using all existing (finite) trust paths between entities. We show in the following of this section, that powers of this adjacency matrix can be evaluated privately in a distributed manner, provided that one disposes of an homomorphic cryptosystem satisfying the homomorphic Properties (1) and (2).

### 3.1 Aggregation of Trust

Consider Alice trusting Bob with a certain trust degree, and Bob trusting Charlie with a certain trust degree. The sequential aggregation of trust formalizes a kind of transitivity to help Alice to make a decision about Charlie, that is based on Bob's opinion. In the following, we first consider that the trust values are given as a pair  $\langle a, b \rangle \in \mathbb{D}^2$ , for  $\mathbb{D}$  a principal ideal ring: for three players  $P_1, P_2$  and  $P_3$ , where  $P_1$  trusts  $P_2$  with trust value  $\langle a, b \rangle \in \mathbb{D}^2$  and  $P_2$  trusts  $P_3$  with trust value  $\langle c, d \rangle \in \mathbb{D}^2$  the associated *sequential aggregation of trust* is a function  $\star : \mathbb{D}^2 \times \mathbb{D}^2 \rightarrow \mathbb{D}^2$ , that computes the trust value over the trust path  $P_1 \xrightarrow{\langle a, b \rangle} P_2 \xrightarrow{\langle c, d \rangle} P_3$  as  $\langle a, b \rangle \star \langle c, d \rangle = \langle ac + bd, ad + bc \rangle$ . Similarly, from Alice to Charlie, there might be several ways to perform a sequential aggregation (several paths with existing trust values). Therefore it is also possible to aggregate these parallel paths with the same measure, in the following way: for two disjoint paths  $P_1 \xrightarrow{\langle a, b \rangle} P_3$  and  $P_1 \xrightarrow{\langle c, d \rangle} P_3$ , the associated *parallel aggregation of trust* is a function  $\boxtimes : \mathbb{D}^2 \times \mathbb{D}^2 \rightarrow \mathbb{D}^2$ , that computes the resulting trust value as:  $\langle a, b \rangle \boxtimes \langle c, d \rangle = \langle a + c - ac, bd \rangle$ . We prove the following Lemma.

**Lemma 1.**  $\langle a, b \rangle$  is invertible for  $\boxtimes$  if and only if ( $b$  is invertible in  $\mathbb{D}$ ) and ( $a = 0$  or  $a - 1$  is invertible).

*Proof.* As  $\langle a + 0 - a \cdot 0, b \cdot 1 \rangle = \langle a, b \rangle$ ,  $\langle 0, 1 \rangle$  is neutral for  $\boxtimes$ . Then, for  $b$  invertible, if  $a = 0$ , then  $\langle 0, b^{-1} \rangle$  is an inverse for  $\langle 0, b \rangle$ . Otherwise, for  $a - 1$  invertible,  $\langle a(a-1)^{-1}, b^{-1} \rangle \boxtimes \langle a, b \rangle = \langle a, b \rangle \boxtimes \langle a(a-1)^{-1}, b^{-1} \rangle = \langle a + a(a-1)^{-1} - a^2(a-1)^{-1}, bb^{-1} \rangle = \langle 0, 1 \rangle$ . Similarly, if  $\langle a, b \rangle \boxtimes \langle c, d \rangle = \langle 0, 1 \rangle$ , then  $bd = 1$  and  $b$  is invertible. Then also  $(a-1)c = a$ . Finally if  $a \neq 0$  and  $a-1$  is a zero divisor, there exists  $\lambda \neq 0$  such that  $\lambda(a-1) = 0$ , thus  $\lambda(a-1)c = 0 = \lambda a$ , but then  $\lambda(a-1) - \lambda a = -\lambda = 0$ . As this is contradictory, the only possibilities are  $a = 0$  or  $a-1$  invertible.  $\square$

### 3.2 Multi-party Private Aggregation

For  $E$  an encryption function, we define the natural morphism on pairs, so that it can be applied to trust values:  $E(\langle a, b \rangle) = \langle E(a), E(b) \rangle$ . We can thus extend homomorphic properties to pairs so that the parallel and sequential aggregation can then be computed homomorphically, provided that one entry is in clear.

**Lemma 2.** With an encryption function  $E$ , satisfying the homomorphic Properties (1) and (2), we have:

$$\begin{aligned} \text{Mul}(E(\langle a, b \rangle); \langle c, d \rangle) &= E(\langle a, b \rangle \star \langle c, d \rangle) \\ &= \langle E(a)^c E(b)^d, E(a)^d E(b)^c \rangle \\ \text{Add}(E(\langle a, b \rangle); \langle c, d \rangle) &= E(\langle a, b \rangle \boxtimes \langle c, d \rangle) \\ &= \langle E(a)E(c)E(a)^{-c}, E(b)^d \rangle \end{aligned}$$

Moreover, those two functions can be computed on an enciphered  $\langle a, b \rangle$ , provided that  $\langle c, d \rangle$  is in clear.

*Proof.* From the homomorphic properties of the encryption functions, we have:  $E(a)^c E(b)^d = E(ac + bd)$ ,  $E(a)^d E(b)^c = E(ad + bc)$ ,  $E(a)E(c)E(a)^{-c} = E(a + c + a(-c))$  and  $E(b)^d = E(bd)$ . For the computation, both right hand sides depend only on ciphered values  $E(a)$ ,  $E(b)$ , and on clear values  $c$  and  $d$  ( $E(c)$  can be computed with the public key, from  $c$ ).  $\square$

This shows, that in order to compute the aggregation of trust privately, the first step is to be able to compute dot-products privately.

## 4 From MPWP to P-MPWP

### 4.1 MPWP description

The *MPWP* protocol (Dolev et al., 2010) is used to securely compute private trust values in an additive reputation system between  $n$  players. Each player

$P_i$  (excepted  $P_1$ , assumed to be the master player) has a private entry  $v_i$ , and  $P_1$  private entries are weights  $u_i$  associated to others players. The goal is to compute a weighted average trust, *i.e.*,  $\sum_{i=2}^n u_i * v_i$ . The idea of *MPWP* is the following: the first player creates a vector  $TV$  containing her private entries ciphered with her own public key using Benaloh's cryptosystem, *i.e.*,  $TV = [E_1(w_2), \dots, E_1(w_n)]$ . Then,  $P_1$  also sends a  $(n-1) \times (n-1)$  matrix  $M$ , with all coefficients initialized to 1 and a variable  $A = 1$ . Once  $(M, TV, A)$  received, each player computes:  $A = A * E_1(u_i)^{v_i} * E_1(z_i)$ , where  $z_i$  is a random value generated by  $P_i$ . At the end, the first player gets  $D_1(A) = \sum_{i=2}^n u_i v_i + z_i$ . Then, the idea is to cut the  $z_i$  values in  $n-1$  positive shares such that  $z_i = \sum_{j=2}^n z_{i,j}$ . Next, each  $z_{i,j}$  is ciphered with the public key of  $P_j$ , the result is stored into the  $i^{th}$  column of  $M$ , and  $M$  is forwarded to the next player. In a second phase, players securely remove the added random values to  $A$ , from  $M = (m_{i,j}) = (E_j(z_{i,j}))$ : each player  $P_j$ , except  $P_1$ , computes her  $PSS_j = \sum_{i=2}^n D_j(m_{i,j}) = \sum_{i=2}^n z_{i,j}$  by deciphering all values contained in the  $j^{th}$  row of  $M$ ; then they send  $\gamma_j = E_1(PSS_j)$  to  $P_1$ , their  $PSS_i$  ciphered with the public key of  $P_1$ . At the end,  $P_1$  retrieves the result by computing  $Trust = D_1(A) - \sum_{j=2}^n D_1(\gamma_j) = D_1(A) - \sum_{j=2}^n PSS_j = D_1(A) - \sum_{j=2}^n \sum_{i=2}^n z_{i,j} = D_1(A) - \sum_{i=2}^n z_i = \sum_{i=2}^n u_i v_i$ .

## 4.2 P-MPWP: A lighter MPWP

*P-MPWP* is a variant of *MPWP* with two main differences: first Paillier's cryptosystem is used instead of Benaloh's, and, second, the overall communications cost is reduced from  $O(n^3)$  to  $O(n^2)$  by sending parts of the matrix only. All steps of *P-MPWP* but those clearly identified in the following are common with *MPWP*, including the players' global settings. Since *P-MPWP* is using a cryptosystem where players can have different modulus, some requirements must be verified in the players' settings. First of all, a bound  $B$  needs to be fixed for the vectors' private coefficients:

$$\forall i, 0 \leq u_i \leq B, 0 \leq v_i \leq B \quad (3)$$

With Benaloh, the common modulus  $M$  must be greater than the dot product, thus at most:

$$(n-1)B^2 < M. \quad (4)$$

Differently, with Paillier, each player  $P_i$  has a different modulus  $N_i$ . Then, by following the *MPWP* protocol steps, at the end of the first round,  $P_1$  obtains  $A = \prod_{i=2}^n E_1(u_i)^{v_i} * E_1(z_i)$ . In order to correctly decipher this coefficient, if the players' values, as well as their random values  $z_i$ , satisfy the bound (3), her modulo  $N_1$  must be greater than  $(n-1)(B^2 + B)$ . For

others players, there is only one deciphering step, at the second round. They received  $(n-1)$  shares all bounded by  $B$ . Hence, their modulus  $N_i$  need only be greater than  $(n-1)B$ . These modulus requirements are summarized in the following lemma:

**Lemma 3.** *Let  $n > 3$  be the number of players. Under the bound (3), if  $\forall i, 0 \leq z_i \leq B$  and if also the modulus satisfy  $(n-1)(B^2 + B) < N_1$  and  $(n-1)B < N_i, \forall i = 2, \dots, n$ , then at the end of *P-MPWP*,  $P_1$  obtains  $S_n = \sum_{i=2}^n u_i * v_i$ .*

Now, the reduction of the communications cost in *P-MPWP*, is made by removing the exchange of the full  $M$  matrix between players. At the  $z_{i,j}$  shares computation, each  $P_i$  directly sends the  $j^{th}$  coefficient to the  $j^{th}$  player instead of storing results in  $T$ . In the end, each player  $P_i$  receives  $(n-1)$  values ciphered with his public key, and he can compute the  $PSS_i$  by deciphering and adding each received values, exactly as in *MPWP*. Thus, each player sends only  $O(n)$  values, instead of  $O(n^2)$ . All remaining steps can be executed as in *MPWP*.

Both Paillier's and Benaloh's cryptosystems provides semantic security, thus the security of *P-MPWP* is not altered. Moreover, since a common bound is fixed *a priori* on private inputs, *P-MPWP* security can be reduced to the one in *MPWP* with the common modulo  $M$  between all players (Michalakis et al., 2012). Finally, since all exploitable (*i.e.*, clear or ciphered with the dedicated key) information exchanged represents a subset of the *MPWP* players' knowledge, if one is able to break *P-MPWP* privacy, then one is also able to break it in *MPWP*.

## 5 A Linear Dot Product Protocol

### 5.1 Overview with Three Players

We first present in Figure 1 our *DSDP<sub>3</sub>* protocol (Distributed and Secure Dot-Product), for 3 players. The idea is that Alice is interested in computing a dimension 3 dot-product  $S = u^T \cdot v$ , between her vector  $u$  and a vector  $v$  whose coefficients are owned by different players. The other players send their coefficients, encrypted, to Alice. Then she homomorphically multiplies each one of these by her  $u_i$  coefficients and masks the obtained  $u_i v_i$  by a random value  $r_i$ . Then the other players can decrypt the resulting  $u_i v_i + r_i$ : with two unknowns  $u_i$  and  $r_i$  they are not able to recover  $v_i$ . Finally the players enter a ring computation of the overall sum before sending it to Alice. Then only, Alice removes her random masks to recover the final dot-product. Since at least two



players have added  $u_2v_2 + u_3v_3$ , there is at least two unknowns for Alice, but a single equation.

We need that after several decryptions and re-encryptions, and removal of the random values  $r_i$ ,  $S$  is exactly  $\sum u_i v_i$ . The homomorphic Properties (1) and (2) only guaranty that  $D(\text{Add}(\text{Mul}(E(v_i); u_i); r_i)) = v_i u_i + r_i \pmod{N_i}$ , for the modulo  $N_i$  of the cryptosystem used by player  $P_i$ . But then these values must be re-encrypted with another player's cryptosystem, potentially with another modulo. Finally Alice also must be able to remove the random values and recover  $S$  over  $\mathbb{Z}$ . On the one hand, if players can share the same modulo  $M = N_i$  for the homomorphic properties then decryptions and re-encryptions are naturally compatible. This is possible for instance in Benaloh's cipher. On the other hand, in a Paillier-like cipher, at the end of the protocol, Alice will actually recover  $S_4 = ((u_2v_2 + r_2) \pmod{N_2} + u_3v_3 + r_3) \pmod{N_3}$ . He can remove  $r_3$ , via  $S_3 = S_4 - r_3 \pmod{N_3}$ , but then  $S_3 = ((u_2v_2 + r_2) \pmod{N_2} + u_3v_3) \pmod{N_3}$ . Now, if vectors coefficients are bounded by say  $B$ , and if the third modulo is larger than the second,  $N_3 > N_2 + B^2$ , the obtained value is actually the exact value over the naturals:  $S_3 = (u_2v_2 + r_2) \pmod{N_2} + u_3v_3$ . Then Alice can remove the second random value, this time modulo  $N_2$ :  $S_2 = (u_2v_2 + u_3v_3) \pmod{N_2}$ , where now  $N_2 > 2B^2$  suffices to recover  $S = S_2 \in \mathbb{N}$ . We generalize this in the following section.

## 5.2 General Protocol with $n$ Players

We give the generalization  $DSDP_n$ , of the protocol of Figure 1 for  $n$  players in Algorithm 4 hereafter. For this protocol to be correct, we use the previously defined bound (3) on the players' private inputs. Then, for  $n$  players, there are two general cases: First, if all the players share the same modulo  $M = N_i$  for all  $i$  for the homomorphic properties, then Alice can also use  $M$  to remove the  $r_i$ . Then, to compute the correct value  $S$ , it is sufficient to satisfy the bound (4). Second, for a Paillier-like cipher, differently, the modulo of the homomorphic properties are distinct. We thus prove the following Lemma 5.

**Lemma 5.** *Under the bound (3), and for any  $r_i$ , let  $M_2 = (u_2v_2 + r_2) \pmod{N_2}$  and  $M_i = (M_{i-1} + u_i v_i + r_i) \pmod{N_i}$ , for  $i = 2 \dots n-1$ . Let also  $S_{n+1} = M_n$  and  $S_i = (S_{i+1} - r_i) \pmod{N_i}$  for  $i = n \dots 2$ . If we have:*

$$\begin{cases} N_{i-1} + (n-i+1)B^2 < N_i, & \text{for all } i = 3 \dots n \\ (n-1)B^2 < N_2 \end{cases} \quad (5)$$

then  $S_2 = \sum_{i=2}^n u_i v_i \in \mathbb{N}$ .

*Proof.* By induction, we first show that  $S_i = M_{i-1} + \sum_{j=i}^n u_j v_j$ , for  $i = n \dots 3$ : indeed  $S_n = (M_n - r_n)$

---

**Algorithm 4**  $DSDP_n$  Protocol: Distributed and Secure Dot-Product of size  $n$

---

**Require:**  $n \geq 3$  players, two vectors  $U$  and  $V$  such that  $P_1$  knows complete vector  $U$ , and each players  $P_i$  knows component  $v_i$  of  $V$ , for  $i = 1 \dots n$ ;

**Require:**  $E_i$  (resp.  $D_i$ ), encryption (resp. decryption) function of  $P_i$ , for  $i = 2 \dots n$ .

**Ensure:**  $P_1$  knows the dot-product  $S = U^T V$ .

```

1: for  $i = 2 \dots n$  do  $\{P_i : c_i = E_i(v_i); P_i \xrightarrow{c_i} P_1\}$ 
2: for  $i = 2 \dots n$  do
3:    $P_1 : r_i \xleftarrow{\$} \mathbb{Z}/N_i\mathbb{Z}$ 
4:    $P_1 : \alpha_i = c_i^{u_i} * E_i(r_i)$  so that  $\alpha_i = E_i(u_i v_i + r_i)$ 
5:  $P_1 \xrightarrow{\alpha_i} P_2$ 
6: for  $i = 2 \dots n-1$  do  $P_1 : \alpha_{i+1} \xrightarrow{P_i} P_i$ 
7:  $P_2 : \Delta_2 = D_2(\alpha_2)$  so that  $\Delta_2 = u_2 v_2 + r_2$ 
8:  $P_2 : \beta_3 = \alpha_3 * E_3(\Delta_2)$  so that  $\beta_3 = E_3(u_3 v_3 + r_3 + \Delta_2)$ ;  $P_2 \xrightarrow{\beta_3} P_3$ 
9: for  $i = 3 \dots n-1$  do
10:   $P_i : \Delta_i = D_i(\beta_i)$  so that  $\Delta_i = \sum_{k=2}^i u_k v_k + r_k$ 
11:   $P_i : \beta_{i+1} = \alpha_{i+1} * E_{i+1}(\Delta_i)$  so that  $\beta_{i+1} = E_{i+1}(u_{i+1} v_{i+1} + r_{i+1} + \Delta_i)$ ;  $P_i \xrightarrow{\beta_{i+1}} P_{i+1}$ 
12:  $P_n : \Delta_n = D_n(\beta_n)$ ;  $P_n : \gamma = E_1(\Delta_n)$ ;  $P_n \xrightarrow{\gamma} P_1$ 
13: return  $P_1 : S = D_1(\gamma) - \sum_{i=1}^{n-1} r_i + u_1 v_1$ .
```

---

$\pmod{N_n} = (M_{n-1} + u_n v_n) \pmod{N_n}$ . But  $M_{n-1}$  is modulo  $N_{n-1}$ , so  $(M_{n-1} + u_n v_n) < N_{n-1} + B^2$ , and then (5) for  $i = n$ , ensures that  $N_{n-1} + B^2 < N_n$  and  $S_n = M_{n-1} + u_n v_n \in \mathbb{N}$ . Then, for  $3 \leq i < n$ ,  $S_i = (S_{i+1} - r_i) \pmod{N_i} = (M_i + \sum_{j=i+1}^n u_j v_j - r_i) \pmod{N_i} = (M_{i-1} + u_i v_i + r_i + \sum_{j=i+1}^n u_j v_j - r_i) \pmod{N_i} = (M_{i-1} + \sum_{j=i}^n u_j v_j) \pmod{N_i}$ , by induction. But (3) enforces that  $M_{i-1} + \sum_{j=i}^n u_j v_j < N_{i-1} + (n-i+1)B^2$  and (5) also ensures the latter is lower than  $N_i$ . Therefore  $S_i = M_{i-1} + \sum_{j=i}^n u_j v_j$  and the induction is proven. Finally,  $S_2 = (S_3 - r_2) \pmod{N_2} = (M_2 + \sum_{j=3}^n u_j v_j - r_2) \pmod{N_2} = (\sum_{j=2}^n u_j v_j) \pmod{N_2}$ . As  $\sum_{j=2}^n u_j v_j < (n-1)B^2$ , by (5) for  $i = 2$ , we have  $S_2 = \sum_{j=2}^n u_j v_j \in \mathbb{N}$ .  $\square$

This shows that the  $DSDP_n$  protocol of Algorithm 4 can be implemented with a Paillier-like underlying cryptosystem, provided that the successive players have increasing modulo for their public keys.

**Theorem 6.** *Under the bounds (3), and under Hypothesis (4) with a shared modulus underlying cipher, or under Hypothesis (5) with a Paillier-like underlying cipher, the  $DSDP_n$  protocol of Algorithm 4 is correct. It requires  $O(n)$  communications and  $O(n)$  encryption and decryption operations.*

*Proof.* First, each player sends his ciphered entry

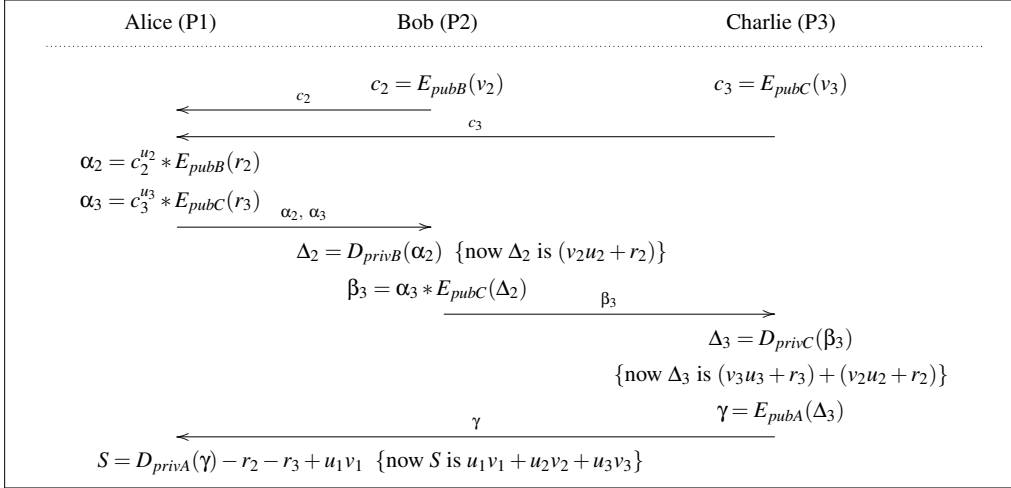


Figure 1:  $DSDP_3$ : Secure dot product of vectors of size 3 with a Paillier-like asymmetric cipher.

to  $P_1$ , then homomorphically added to random values,  $r_i$ . Then,  $P_i$  ( $i \geq 2$ ) deciphers the message received by  $P_{i-1}$  into  $\Delta_i$ . By induction, we obtain  $\Delta_i = \sum_{k=2}^i u_k v_k + r_k$ . This value is then re-encrypted with next player's key and the next player share is homomorphically added. Finally,  $P_1$  just has to remove all the added randomness to obtain  $S = \Delta_n - \sum_{i=2}^n r_i + u_1 v_1 = \sum_{i=1}^n u_i v_i$ . For the complexity, the protocol needs  $n - 1$  encryptions and communications for the  $c_i$ ;  $2(n - 1)$  homomorphic operations on ciphers and  $n - 1$  communications for the  $\alpha_i$ ;  $n - 1$  decryptions for the  $\Delta_i$ ;  $n - 1$  encryptions, homomorphic operations and communications for the  $\beta_i$ ; and finally one encryption and one communication for  $\gamma$ . Then  $P_1$  needs  $O(n)$  operations to recover  $S$ .  $\square$

## 6 Security of the $DSDP$ protocol

We study the security of  $DSDP_n$  using both mathematical proofs and automated verifications. We first demonstrate the security of the protocol for *semi-honest* adversaries. Then we incrementally build its security helped by attacks found by ProVerif, an automatic verification tool for cryptographic protocols.

### 6.1 Security Proofs

The standard security definition in MPC models (Lindell, 2009) covers actually many security issues, such as correctness, inputs independence, privacy, etc. We first prove that under this settings, computation of the dot product is safe.

**Lemma 7.** For  $n \geq 3$ , the output obtained after computing a dot product where one player owns complete

vector  $U$ , and where each coefficient  $v_i$  of the second vector  $V$  is owned by the player  $P_i$ , is safe.

*Proof.* After executing  $DSDP_n$  with  $n \geq 3$ ,  $P_1$  received the dot product of  $U$  and  $V$ . Therefore, it owns only one equation containing  $(n - 1)$  unknown values (coefficients from  $v_2$  to  $v_n$ ). Then, he cannot deduce other players' private inputs.  $\square$

Then, proving the security relies on a comparison between a real-world protocol execution and an ideal one. The latter involves an hypothetical trusted third party ( $TTP$ ) which, knowing only the players' private inputs, returns the correct result to the correct players. The protocol is considered secure if the players' views in the ideal case cannot be distinguished from the real ones. Views of a player  $P_i$  (denoted  $View_{P_i}$ ) are defined as distributions containing: the players' inputs (including random values), the messages received during a protocol execution and the outputs. The construction of the corrupted players' view in the ideal world is made by an algorithm called *Simulator*.

**Definition 8.** In the presence of a set  $C$  of semi-honest adversaries with inputs set  $X_C$ , a protocol  $\Pi$  securely computes  $f : ([0, 1]^*)^m \rightarrow ([0, 1]^*)^m$  (and  $f_C$  denotes the outputs of  $f$  for each adversaries in  $C$ ) if there exists a probabilistic polynomial-time algorithm  $Sim$ , such that:  $\{Sim(C, \{X_C\}, f_C(X))\}_{X \in ([0, 1]^*)^m}$  is computationally indistinguishable from  $\{C, \{View_{P_i}^\Pi\}_{P_i \in C}\}$ .

For  $DSDP_n$ , it is secure only if  $C$  is reduced to a singleton, i.e. if only one player is corrupted.

**Lemma 9.** By assuming the semantic security of the cryptosystem  $E$ , for  $n \geq 3$ ,  $DSDP_n$  is secure against one semi-honest adversary.

*Proof.* We assume that the underlying cryptosystem  $E$  is semantically secure (IND-CPA secure). First, we suppose that only  $P_1$  is corrupted. His view, in a real execution of the protocol, is  $View_{P_1} = \{U, R, \gamma, S, A, B, C\}$ , where  $U = \{u_i\}_{1 \leq i \leq n}$ ,  $R = \{r_i\}_{1 \leq i \leq n}$ ,  $A = \{\alpha_i\}_{2 \leq i \leq n}$ ,  $B = \{\beta_i\}_{3 \leq i \leq n-1}$  and  $C = \{c_i\}_{2 \leq i \leq n}$ . Now,  $Sim_1$  is the simulator for  $P_1$  in the ideal case, where a simulated value  $x$  is denoted  $x'$ : by definition,  $P_1$ 's private entries (vectors  $U$  and  $R$ ) are directly accessible to  $Sim_1$ , along with the output  $S$ , sent by the  $TTP$ .  $Sim_1$  starts by generating  $n - 2$  random values, and then ciphers them using the corresponding public keys: this simulates the  $c'_i$  values. Then, using the provided  $r_i$  and  $u_i$  with the associated  $c'_i$  and  $P_i$ 's public key,  $Sim_1$  computes:  $\alpha'_i = c_i^{u_i} * E_i(r_i), 2 \leq i \leq n$ . Next, the simulation of  $B'$  is done by ciphering random values with the appropriate public key. The  $\gamma'$  value is computed using  $R$  along with the protocol output  $S$ :  $\gamma' = E_1(S + \sum_{i=2}^{n-2} r_i + u_1 v_1)$ . In the end, the simulator view is  $View_{Sim_1} = \{U, R, \gamma', S, A', B', C'\}$ . If an adversary is able to distinguish any ciphered values (e.g.  $C'$  from  $C$  and thus  $A'$  from  $A$ ), hence he is able to break the semantic security of the underlying cryptographic protocol. This is assumed impossible. Moreover, since the remaining values are computed as in a real execution,  $P_1$  is not able to distinguish  $View_{P_1}$  from  $View_{Sim_1}$ . Second, we suppose that a player  $P_i, i \geq 2$  is corrupted and denote by  $Sim_i$  the simulator in this case. Since the role played by each participant is generic, (except for  $P_n$ , which only differs by his computation of  $\gamma$  instead of  $\beta_{n+1}$ ), the simulators are easily adaptable. During a real protocol execution, the view of  $P_i$  is  $View_{P_i} = \{v_i, A, B, C, \gamma, \Delta_i\}$ . Simulating the values also known to  $P_1$  is similar, up to the used keys. Hence, the simulation of  $A', B', \gamma', C'$  (except  $c_i$ ) is made by ciphering random values using the adequate public key.  $c_i$  is ciphered using  $v_i$  and the public key of  $P_i$ . For  $\Delta'_i$ , the simulator  $Sim_i$  has to forward the random value previously chosen to be ciphered as  $\alpha_i$ . Indistinguishability is based on the semantic security of  $E$  (for  $A, B, C$  and  $\gamma$ ) and on the randomness added by  $P_1$  (and thus unknown by  $P_i$ ). Then,  $\Delta'_i$  is computationally indistinguishable from the real  $\Delta_i$ . Hence,  $View_{P_i}$  and  $View_{S_i}$  are indistinguishable and  $DSDP_n$  is secure against one semi-honest adversary.  $\square$

## 6.2 Automated Verification

Alongside mathematical proofs, we use an automatic protocol verification tool to analyze the security of the protocol. Among existing tools, we use ProVerif (Blanchet, 2001; Blanchet, 2004). It allows users to add their own equational theories to model a

large class of protocols. In our case, we model properties of the underlying cryptosystem including addition and multiplication. Sadly, verification of protocol in presence of homomorphic function over abelian groups theory has been proven undecidable (Delaune, 2006). Moreover, as showed in (Lafourcade and Puys, 2015), some equational theories such as Exclusive-Or can already outpace the tool's capacities. Thus we have to provide adapted equational theories to be able to obtain results with the tool. We modeled the application of Pailler's or shared modulus encryption properties on  $\alpha_i$  messages that Bob receives as follows:

$$(i). \forall u, v, r, k, \text{bob}(E_k(r), u, E_k(v)) = E_k(uv + r)$$

This property allows Bob to obtain  $u_2 v_2 + r_2$  from  $\alpha_2$ . This also allows an intruder to simulate such calculus and impersonate Bob. We also model:

$$(ii). \beta_3 \text{ by } \forall u, v, r, x, y, z, k, \text{charlie}(E_k(uv + r), E_k(xy + z)) = E_k(uv + xy + r + z)$$

$$(iii). \beta_4 \text{ by } \forall u, v, r, x, y, z, a, b, c, k, \text{dave}(E_k(uv + xy + r + z), E_k(ab + c)) = E_k(uv + xy + ab + r + z + c)$$

In the following, we use ProVerif to prove the security of our protocols under the abstraction of the functionalities given in our equational theory. ProVerif discovers some attacks in presence of active intruder. We then propose some countermeasures. The limits of ProVerif are reached and it does not terminate. The associated source files are available in a web-site: [matmuldistrib.forge.imag.fr](http://matmuldistrib.forge.imag.fr).

**Analysis in case of a passive adversary.** Using these equational theories on the protocol described in Figure 1, we verify it in presence of a passive intruder. Such adversary is able to observe all the traffic of the protocol and tries to deduce secret information of the messages. This corresponds to a ProVerif intruder that only listens to the network and does not send any message. By default, this intruder does not possess the private key of any agent and thus does not belong to the protocol. To model a *semi-honest* adversary as defined in Section 2, we just give secret keys of honest participants to the passive intruder knowledge in ProVerif. Then the tool proves that all secret terms cannot be learned by the intruder for any combinations of leaked key. This confirms the proofs given in Section 6.1 against the *semi-honest* adversaries.

**Analysis in case of malicious adversary.** The *malicious* adversary described in Section 2 is an active intruder that controls the network and knows a private key of a compromised honest participant. Modeling this adversary in ProVerif, we are able to spot the two following attacks and give some countermeasures:

- (i) Only the key of Alice is compromised and the countermeasure uses proofs of knowledge.



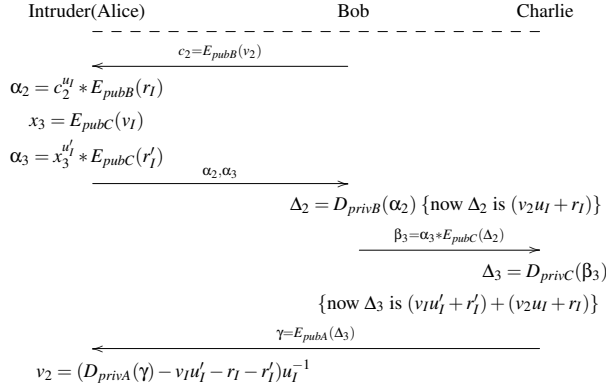


Figure 2: Attack on the secrecy of  $v_2$

(ii) Only the key of Charlie is compromised and the countermeasure uses signatures.

In the rest of the section, we present these two points. In the Section 7.2, we also give a solution called *random ring* for the case where both keys of Alice and Charlie are compromised.

(i) *The key of Alice is compromised.* An attack on the secrecy of  $v_2$ , the secret generated by Bob, is then presented in Figure 2.

The malicious adversary usurps Alice and replaces all the  $\alpha_i$  messages, arriving from the other agents, with one message she generated, except one message, denoted  $c_2$  in Figure 2. He lets the protocol end normally and obtains a term where only  $v_2$  is unknown. He learns  $v_2$ . If the key of Alice ( $P_1$ ) is compromised, ProVerif also finds an attack on any of the other players secrecy. Suppose, w.l.o.g, that  $P_2$  is the target,  $P_1$  replaces each  $\alpha_i$  except  $\alpha_2$  by ciphers  $E_i(x_i)$  where  $x_i$  are known to him.  $x_i = 0$  could do for instance ( $x_i = 0v_i + r_i$  also), since after completion of the protocol,  $P_1$  learns  $u_2 v_2 + r_2 + \sum_{i=3}^n x_i$ , where the  $u_i$  and  $r_i$  are known to him. Therefore,  $P_1$  learns  $v_2$ . Note also that similarly, for instance,  $\alpha_2 = 1v_2 + 0$  and  $x_3 = v_3$  could also reveal  $v_2$  to  $P_3$ . *Counter measure:* this attack, and more generally attacks on the form of the  $\alpha_i$  can be counteracted by zero-knowledge proofs of knowledge.  $P_1$  has to prove to the other players that  $\alpha_i$  is a non trivial affine transform of their secret  $v_i$ . For this we use a variant of a proof of knowledge of a discrete logarithm (Chaum et al., 1986) given in Figure 3.

In the Protocol 4, this proof of a non trivial affine transform applies as is to  $\alpha_2$  with  $\mu_2 = g^{u_2}$ ,  $\rho_2 = g^{r_2}$  so that the check of  $P_2$  is  $\delta_2 = g^{\Delta_2} \stackrel{?}{=} \mu_2^{v_2} \rho_2$ . Differently, for the subsequent players, the  $\delta_{i-1} = g^{\Delta_{i-1}}$  used to test must be forwarded: indeed the subsequent players have to check in line 10 that  $\Delta_i = u_i v_i + r_i + \Delta_{i-1}$ . Thus with  $P_1$  providing  $\mu_i = g^{u_i}$ ,  $\rho_i = g^{r_i}$  and  $P_{i-1}$  providing  $\delta_{i-1}$ , the check of player  $P_i$  ends with  $\delta_i =$

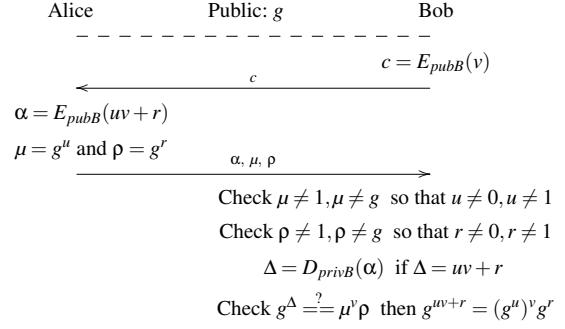


Figure 3: Proof of a non trivial affine transform

$g^{\Delta_i} \stackrel{?}{=} \mu_i^{v_i} \rho_i \delta_{i-1}$ . As for proofs of knowledge of discrete logarithm, secrecy of our proof of non trivial affine transform is guaranteed as long as the discrete logarithm is difficult. The overhead in the protocol, in terms of communications, is to triple the size of the messages from  $P_1$  to  $P_i$ , with  $\alpha_i$  growing to  $(\alpha_i, \mu_i, \rho_i)$ , and to double the size of the messages from  $P_i$  to  $P_{i+1}$ , with  $\beta_i$  growing to  $(\beta_i, \delta_i)$ . In terms of computations, it is also a neglectible linear global overhead.

(ii) *The key of Charlie is compromised.* There ProVerif finds another attack on the secrecy of  $v_2$ . This time the key of Charlie is compromised and the malicious adversary blocks all communications to and from Alice who is honest. The adversary performs the same manipulation on the  $\alpha_i$  terms which are directly sent to Bob. Thus, this attack becomes feasible since the adversary knows the terms  $u_2, u_3, r_2, r_3$  and  $v_3$  that he generated and  $\Delta_3 = (v_2 u_2 + r_2) + (v_3 u_3 + r_3)$  using the private key of Charlie. Such an attack relies on the fact that Bob has no way to verify if the message he receives from Alice has really been sent by Alice. This can be avoided using cryptographic signatures.

This attack can be generalized to any number of participants. The attack needs the adversary to know the key of Alice (since she is the only one to know the  $u_i$  and  $r_i$  values thanks to the signatures). Then, to obtain the secret value of a participant  $P_i$ , the key of participants  $P_{i-1}$  and  $P_{i+1}$  are also needed:

- (i).  $P_{i-1}$  knows  $\Delta_{i-1} = (u_2 v_2 + \dots + u_{i-1} v_{i-1} + r_2 + \dots + r_{i-1})$ .
- (ii).  $P_{i+1}$  knows  $\Delta_{i+1} = (u_2 v_2 + \dots + u_{i-1} v_{i-1} + u_i v_i + u_{i+1} v_{i+1} + r_2 + \dots + r_{i-1} + r_i + r_{i+1})$ .

Thus, by simplifying  $\Delta_{i-1}$  and  $\Delta_{i+1}$ , the malicious adversary obtains  $u_i v_i + u_{i+1} v_{i+1} + r_i + r_{i+1}$  where he can remove  $u_{i+1}, v_{i+1}, r_i, r_{i+1}$  and  $u_i$  to obtain  $v_i$ . For more than three participants, we see in Section 7.2 that these kinds of threats can be diminished if the protocol is replayed several times in random orders.

## 7 Parallel Communications

In order to speed up the overall process, we show that we can cut each dot-product into blocks of 2 or 3 coefficients. On the one hand, the overall volume of communications is unchanged, while the number of rounds is reduced from  $n$  to a maximum of 5. On the other hand, semantic security is dropped, but we will see at the end of this section that by simply repeating the protocol with a wiretap mask it is possible to make the probability of breaking the protocol negligible.

An application of the  $DSDP_n$  protocol is the computation of matrix multiplication. In this case, instead of knowing one vector, each player  $P_i$  owns two rows,  $A_i$  and  $B_i$ , one of each  $n \times n$  matrices  $A$  and  $B$ . At the end, each  $P_i$  learns a row  $C_i$  of the matrix  $C = AB$ . In order to compute the matrix product, it is therefore natural to parallelize  $DSDP_n$ : each dot-product is cut into blocks of 2 or 3 coefficients. Indeed, scalar product between three players (resp. four) involves two (resp. three) new coefficients in addition to the ones already known by  $P_i$ . For  $P_1$ , the idea is to call  $DSDP_3$  on the coefficients  $u_1, v_1$  and  $u_2, u_3$  of  $P_1$ , and  $v_2, v_3$  of  $P_2$  and  $P_3$ . Then  $P_1$  knows  $s = u_1v_1 + u_2v_2 + u_3v_3$ .  $P_1$  can then continue the protocol with  $P_4$  and  $P_5$ , using  $(s, 1)$  as his first coefficient and  $u_4, u_5$  to be combined with  $v_4, v_5$ , etc.  $P_1$  can also launch the computations in parallel. Then  $P_1$  adds his share  $u_1v_1$  only after all the computations. For this it is sufficient to modify line 13 of  $DSDP_n$  as:  $P_1 : S = D_1(\gamma) - \sum_{i=1}^{n-1} r_i$ . This is given as the  $ESDP_n$  protocol variant in Algorithm 10.

---

**Algorithm 10**  $ESDP_n$  Protocol: External Secure Dot-Product of size  $n$

**Require:**  $n+1$  players,  $P_1$  knows a coefficient vector  $U \in \mathbb{F}^n$ , each  $P_i$  knows components  $v_{i-1}$  of  $V \in \mathbb{F}^n$ , for  $i = 2 \dots n+1$ .

**Ensure:**  $P_1$  knows  $S = U^T V$ .

**return**  $DSDP_{n+1}(P_1 \dots P_{n+1}, [0, U], [0, V])$ .

---

### 7.1 Partition in Pairs or Triples

Depending on the parity of  $n$ , and since  $\gcd(2, 3) = 1$ , calls to  $ESDP_2$  and  $ESDP_3$  are sufficient to cover all possible dot-product cases, as shown in protocol  $PDSMM_n$  of Algorithm 11. The protocol is cut in two parts. The loop allows us to go all over coefficients by block of size 2. In the case where  $n$  is even, a block of 3 coefficients is treated with an instance of  $ESDP_3$ . In terms of efficiency and depending on the parity of  $n$ ,  $ESDP_2$  is called  $\frac{n-1}{2}$  or  $\frac{n}{2} - 2$  times, and  $ESDP_3$  is called 0 or 1 times.

---

**Algorithm 11**  $PDSMM_n$  Protocol: Parallel Distributed and Secure Matrix Multiplication

---

**Require:**  $n$  players, each player  $P_i$  knows rows  $A_i$  and  $B_i$  of two  $n \times n$  matrices  $A, B$ .

**Ensure:** Each player  $P_i$  knows row  $i$  of  $C = AB$ .

```

1: for Each row:  $i=1 \dots n$  do
2:   for Each column:  $j=1 \dots n$  do
3:      $s \leftarrow a_{i,i}b_{i,j}$ 
4:     if  $n$  is even then
5:        $k_1 \leftarrow (i-1) \bmod n + 1$ ;  $k_2 \leftarrow (i-2) \bmod n + 1$ ;  $k_3 \leftarrow (i-3) \bmod n + 1$ ;
6:        $s \leftarrow s + ESDP_3(P_i, [P_{k_3}, P_{k_2}, P_{k_1}], [a_{i,k_3}, a_{i,k_2}, a_{i,k_1}], [b_{k_3,j}, b_{k_2,j}, b_{k_1,j}])$ 
7:        $t \leftarrow \frac{n-4}{2}$ 
8:     else
9:        $t \leftarrow \frac{n-1}{2}$ 
10:    for  $h = 1 \dots t$  do
11:       $k_1 \leftarrow (i+2h-1) \bmod n + 1$ ;  $k_2 \leftarrow (i+2h) \bmod n + 1$ ;
12:       $s \leftarrow s + ESDP_2(P_i, [P_{k_1}, P_{k_2}], [a_{i,k_1}, a_{i,k_2}], [b_{k_1,j}, b_{k_2,j}])$ 
13:     $c_{i,j} \leftarrow s$ 

```

---

**Theorem 12.** *The  $PDSMM_n$  Protocol in Algorithm 11 is correct. It runs in less than 5 parallel communication rounds.*

*Proof.* Correctness means that at the end, each  $P_i$  has learnt row  $C_i$  of  $C = AB$ . Since the protocol is applied on each rows and columns, let us show that for a row  $i$  and a column  $j$ , Algorithm 11 gives the coefficient  $c_{ij}$  such that  $c_{ij} = \sum_{k=1}^n a_{ik} * b_{kj}$ . First, the  $k_i$  coefficients are just the values  $1 \dots (i-1)$  and  $(i+1) \dots n$  in order. Then, the result of any  $ESDP_2$  step is  $a_{i,k_1}b_{k_1,j} + a_{i,k_2}b_{k_2,j}$  and the result of the potential  $ESDP_3$  step is  $a_{i,k_3}b_{k_3,j} + a_{i,k_2}b_{k_2,j} + a_{i,k_1}b_{k_1,j}$ . Therefore accumulating them in addition of  $a_{i,i} * b_{i,j}$  produces as expected  $c_{ij} = \sum_{k=1}^n a_{ik} * b_{kj}$ .

Now for the number of rounds, for all  $i$  and  $j$ , all the  $ESDP$  calls are independent. Therefore, if each player can simultaneously send and receive multiple data we have that: in parallel,  $ESDP_2$ , like  $DSDP_3$  in Figure 1, requires 4 rounds with a constant number of operations: one round for the  $c_i$ , one round for the  $\alpha_i$ , one round for  $\beta_3$  and one round for  $\gamma$ . As shown in Algorithm 4,  $ESDP_3$ , like  $DSDP_4$ , requires only a single additional round for  $\beta_4$ .  $\square$

### 7.2 Random Ring Order Mitigation

We have previously seen that if the first player of a dot-product cooperates with the third one she can always recover the second player private value. If the

first player cooperates with two well placed players she can recover the private value of a player in between. In the trust evaluation setting every malicious player plays the role of the first player in its row and therefore as soon as there is a collaboration, there is a risk of leakage. To mitigate this cooperation risk, our idea is to repeat the dot product protocol in random orders, except for the first player. To access a given private value, the malicious adversaries have to be well placed in *every* occurrence of the protocol. Therefore if their placement is chosen uniformly at random the probability that they recover some private value diminishes with the number of occurrences. In practice, they use a pseudo, but unpredictable, random generator to decide their placement: as each of them has to know their placement, they can for instance use a cryptographic hash function seeded with the alphabetical list of the players distinguished names, with the date of the day and with random values published by each of the players. We detail the overall procedure only for one dot-product, within the  $PDSMM_n$  protocol. Each player except the first one masks his coefficient  $v$  as in a simple wiretap channel (Ozarow and Wyner, 1984), as sketched in Algorithm 13.

---

**Algorithm 13** Wiretap repetition of the dot-product

---

- 1: The players agree on  $d$  occurrences.
  - 2: Each player computes his placement order in each occurrence of the protocol from the cryptographic hash function.
  - 3a: With a shared modulus cryptosystem, the players should share a common modulo  $M$  satisfying Hypothesis (4). In the first occurrence, each player  $P_j$  then masks his private input coefficient  $v_j$  with  $d-1$  random values  $\lambda_{j,i} \in \mathbb{Z}/M\mathbb{Z}$ :  $v_j - \sum_{i=2}^d \lambda_{j,i}$ .
  - 3b: With a Paillier-like cryptosystem, the players choose their moduli according to Hypothesis (5), where  $B^2$  is replaced by  $dB^2$ , in groups of size  $n=4$  (the requirements of (5) on the moduli are somewhat sequential, but can be satisfied independently if each modulo is chosen in a distinct interval larger than  $3dB^2$ ). Then, in the first occurrence, each player  $P_j$  masks his private input coefficient  $v_j$  with  $d-1$  random values  $0 \leq \lambda_{j,i} < B$ :  $v_j + \sum_{i=2}^d (B - \lambda_{j,i}) < dB$ .
  - 4: Then for each subsequent occurrence, each player replaces its coefficient by one of the  $\lambda_{j,i}$ .
  - 5: In the end, the first player has gathered  $d$  dot-products and just needs to sum them in order to recover the correct one.
- 

**Theorem 14.** *Algorithm 13 correctly allows the first player to compute the dot-product.*

*Proof.* First, in a shared modulus setting, after the first occurrence, Alice ( $P_1$ ) gets  $S_1 = \sum_{j=2}^n u_j (v_j - \sum_{i=2}^d \lambda_{j,i})$ . Then in the following occurrences, Alice gets  $S_i = \sum_{j=2}^n u_j \lambda_{j,i}$ . Finally she computes  $\sum_{i=1}^d S_i = \sum_{j=2}^n u_j v_j$ . Second, similarly, in a Paillier-like setting, after the first occurrence, Alice recovers  $S_1 = \sum_{j=2}^n u_j (v_j + \sum_{i=2}^d (B - \lambda_{j,i}))$ . Then in the following occurrences, Alice gets  $S_i = \sum_{j=2}^n u_j \lambda_{j,i}$ . Finally she computes  $\sum_{i=1}^d S_i - (d-1)B(\sum_{j=2}^n u_j) = \sum_{j=2}^n u_j (v_j + (d-1)B) - (d-1)Bu_j = \sum_{j=2}^n u_j v_j$ .  $\square$

We give now the probability of avoiding attacks in the case when  $n = 2t + 1$ , but the probability in the even case should be close.

**Theorem 15.** *Consider  $n = 2t + 1$  players, grouped by 3, of which  $k \leq n - 2$  are malicious and cooperating, including the first one Alice. Then, it is on average sufficient to run Algorithm 13 with  $d \leq 2 \ln(\min\{k-1, n-k, \frac{n-1}{2}\}) (1 + \frac{k-1}{n-k-1})$  occurrences, to prevent the malicious players from recovering any private input of the non malicious ones.*

*Proof.* The idea is that for a given private input of a non malicious player Bob, to be revealed to Alice, Bob needs to be placed between cooperating malicious adversaries at each occurrence of the protocol. If there is only one non malicious player, then nothing can be done to protect him. If there is 2 non malicious, they are safe if they are together one time, this happens with probability  $\frac{1}{n-2}$ , and thus on average after  $n-2$  occurrences. Otherwise,  $PDSMM_n$  uses  $t = \frac{n-1}{2}$  groups of 3, including Alice. Thus, each time a group is formed with one malicious and one non malicious other players, Alice can learn the private value of the non malicious player. Now, after any occurrence, the number  $a$  of attacked players is less than the number of malicious players minus 1 (for Alice) and obviously less than the number of non malicious players:  $0 \leq a < \min\{k-1, n-k\}$ . Thus let  $b = k-1-a$  and  $c = n-k-a$ . In the next occurrence, the probability of saving at least one more non malicious is  $\frac{a(a-1+c)(n-3)!}{(n-1)!} \frac{n-1}{2} = \frac{a(a-1+c)}{2(n-2)} = \frac{a(n-k-1)}{2(n-2)}$ , so that the average number of occurrences to realize this is  $\mathbb{E}_{n,k}(a) = \frac{2(n-2)}{a(n-k-1)}$ . Thus,  $T_{n,k}(a)$ , the average number of occurrences to save all the non malicious players, satisfies  $T_{n,k}(a) \leq \mathbb{E}_{n,k}(a) + T_{n,k}(a-1) \leq \sum_{i=a}^3 \mathbb{E}_{n,k}(i) + T_{n,k}(2) = (\sum_{i=a}^3 \frac{1}{i}) \frac{2(n-2)}{n-k-1} + T_{n,k}(2)$ . With 2 attacked and  $c$  saved,  $T_{n,k=n-c-2}(2) = \frac{n-2}{c+1}$  so that  $T_{n,k}(a) \leq (H_a - \frac{3}{2}) \frac{2(n-2)}{n-k-1} + \frac{n-2}{n-k-1}$ , where bounds on the Harmonic numbers give  $H_a \leq \ln a$  (see, e.g., (Batir,

2011)) and since  $a \leq k - 1$  and  $a \leq n - k$ , this shows also that  $2a \leq n - 1$ . Therefore,  $T_{n,k}(a) \leq 2 \ln(\min\{k - 1, n - k, \frac{n-1}{2}\}) \frac{n-2}{n-k-1}$ .  $\square$

For instance, if  $k$ , the number of malicious insiders, is less than the number of non malicious ones, the number of repetitions sufficient to prevent any attack is on average bounded by  $O(\log k)$ . To guaranty a probability of failure less than  $\epsilon$ , one needs to consider also the worst case. There, we can have  $k = n - 2$  malicious adversaries and the number of repetitions can grow to  $n \ln(1/\epsilon)$ :

**Proposition 16.** *With  $n = 2t + 1$ , the number  $d$  of random ring repetitions of Algorithm 13 to make the probability of breaking the protocol lower than  $\epsilon$  satisfies  $d < n \ln(1/\epsilon)$  in the worst case.*

*Proof.* There are at least 2 non-malicious players, otherwise the dot-product reveals the secrets in any case. Any given non-malicious player is safe from any attacks if in at least one repetition he was paired with another non-malicious player. In the worst case,  $k = n - 2$  players are malicious and the latter event arises with probability  $(1 - \frac{1}{n-1})^d$  for  $d$  repetitions. If  $d \geq n(\ln(\epsilon^{-1}))$ , then  $d > (n-1)(-\ln \epsilon) > \frac{\ln \epsilon}{\ln(1 - \frac{1}{n-1})}$ , which shows that  $(1 - \frac{1}{n-1})^d < \epsilon$ .  $\square$

Overall, the wiretap variant of Algorithm 13 can guaranty any security, at the cost of repeating the protocol. As the number of repetitions is fixed at the beginning by all the players, all these repetitions can occur in parallel. Therefore, the overall volume of communication is multiplied by the number of repetitions, while the number of rounds remains constant. This is summarized in Table 1 and Figure 4, for the average (Theorem 15) and worst (Proposition 16) cases of Algorithm 13, and where the protocols of the previous sections are also compared.

Table 1: Communication complexities

Protocol	Volume	Rounds	Paillier
MPWP	$O(n^3)$	$O(n)$	$\times$
P-MPWP (§ 4)	$n^{2+o(1)}$	$O(n)$	$\checkmark$
Alg. 13 (Wiretap)	$n^{2+o(1)} \ln(\frac{1}{\epsilon})$	5	$\checkmark$
Alg. 4 (DSDP <sub>n</sub> )	$n^{1+o(1)}$	$O(n)$	$\checkmark$
Alg. 11 (PDSMM <sub>n</sub> )	$n^{1+o(1)}$	5	$\checkmark$
Alg. 13 (Average)	$n^{1+o(1)}$	5	$\checkmark$

On the one hand, we see in Figure 4 that quadratic protocols, with homomorphic encryption, are not usable for a realistic large group of players (trust aggregation could be used for instance by certificate authorities, and there are several hundreds of those in current operating systems or web browsers). On the other

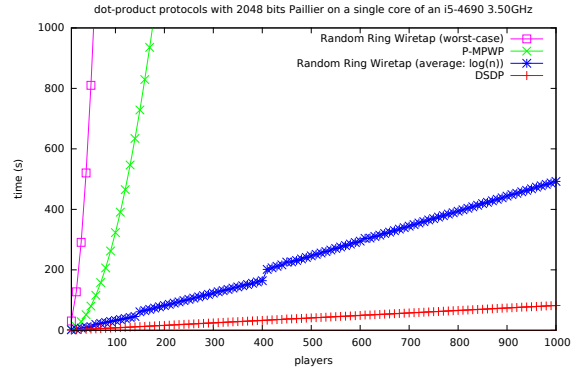


Figure 4: Quadratic and linear protocols timings

hand, quasi linear time protocols present good performance, while preserving some reasonable security properties: the average wiretap curve is on average sufficient to prevent any attack and still has a quasi linear asymptotic behavior. The steps in this curve are the rounding of  $\log(n)$  to the next integer and correspond to one more random ring wiretap round.

## 8 Conclusion: MPC of Trust

We now come back to the aggregation of trust. As shown in Section 3, the first step is to reduce the computation to that of dot-products. We show how to fully adapt the protocol of Section 5 to the evaluation of trust values with parallel and sequential aggregations:

**Corollary 17.** *The protocol DSDP of Algorithm 4 can be applied on trust values, provided that the random values  $r_i$  are invertible for  $\star$ .*

*Proof.*

- $u_i, v_i, r_i, c_i, \alpha_i, \beta_i, \Delta_i, \gamma$  are now couples;
- Encryption and decryption ( $E(v_i), D(\beta_i), E(\Delta_i), E(\gamma)$ , etc.) now apply on couples, using the morphism  $E(\langle a, b \rangle) = \langle E(a), E(b) \rangle$ ;
- $\alpha_i$  is  $E((u_i \star v_i) \star r_i) = \text{Add}(\text{Mul}(E(v_i); u_i); r_i)$ , and can still be computed by  $P_1$ , since  $c_i = E(v_i)$  and  $u_i$  and  $r_i$  are known to him;
- Similarly,  $\beta_i = E(\alpha_i \star \Delta_i) = \text{Add}(E(\alpha_i); \Delta_i)$ .
- Finally, as  $\star$  is commutative,  $S$  is recovered by adding the inverses for  $\star$  of the  $r_i$ .  $\square$

From (Dumas and Hossayni, 2012, Definition 11), the  $d$ -aggregation of trust is a dot-product but slightly modified to not include the value  $u_1 v_1$ . Therefore at line 3, in the protocol of Algorithm 11, it suffices to set  $s$  to the neutral element of  $\star$  (that is  $s \leftarrow \langle 0, 1 \rangle$ , instead of  $s \leftarrow a_{i,j} b_{i,j}$ ).



There remains to encode trust values that are proportions, in  $[0, 1]$ , into  $\mathbb{D} = \mathbb{Z}/N\mathbb{Z}$ . With  $n$  participants, we use a fixed precision  $2^{-p}$  such that  $2^{n(2p+1)} < N \leq 2^{n(2(p+1)+1)}$  and round the trust coefficients to  $\lfloor x2^p \rfloor \bmod N$  from  $[0, 1] \rightarrow \mathbb{D}$ . Then the dot-product can be bounded as follows:

**Lemma 18.** *If each coefficient of the  $u_i$  and  $v_i$  are between 0 and  $2^p - 1$ , then the coefficients of  $S = \star_{i=1}^n (u_i \star v_i)$  are bounded by  $2^{n(2p+1)}$  in absolute value.*

*Proof.* For all  $u, v$ , the coefficients of  $(u \star v)$  are between 0 and  $(2^p - 1)(2^p - 1) + (2^p - 1)(2^p - 1) = 2^{2p+1} - 2^{p+2} + 2 < 2^{2p+1} - 1$  for  $p$  a positive integer. Then, by induction, when aggregating  $k$  of those with  $\star$ , the absolute values of the coefficients remain less than  $2^{k(2p+1)} - 1$ .  $\square$

Therefore, with  $N$  an 2048 bits modulus and  $n \leq 4$  in the *ESDP* protocols of Algorithm 11, Lemma 18 allows a precision close to  $2^{-255} \approx 10^{-77}$ .

In conclusion, we provide an efficient and secure protocol *DSDP<sub>n</sub>* to securely compute dot products (against semi-honest adversary) in the MPC model, with unusual data division between  $n$  players. It can be used to perform a private matrix multiplication and also be adapted to securely compute trust aggregation between players.

## REFERENCES

- Amirbekyan, A. and Estivill-Castro, V. (2007). A new efficient privacy-preserving scalar product protocol. In *AusDM 2007*, volume 70 of *CRPIT*, pages 209–214.
- Batir, N. (2011). Sharp bounds for the psi function and harmonic numbers. *Mathematical inequalities and applications*, 14(4).
- Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC'88*. ACM.
- Benaloh, J. (1994). Dense probabilistic encryption. In *SAC'94*.
- Bendlin, R., Damgård, I., Orlandi, C., and Zakarias, S. (2011). Semi-homomorphic encryption and multi-party computation. In *EUROCRYPT'11*, LNCS.
- Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules. In *IEEE CSFW'01*.
- Blanchet, B. (2004). *Cryptographic Protocol Verifier User Manual*.
- Chaum, D., Evertse, J., van de Graaf, J., and Peralta, R. (1986). Demonstrating possession of a discrete logarithm without revealing it. In *CRYPTO'86*.
- Damgård, I., Pastro, V., Smart, N., and Zakarias, S. (2012). Multiparty computation from somewhat homomorphic encryption. In *CRYPTO'12*, LNCS.
- Delaune, S. (2006). An undecidability result for agh. *Theor. Comput. Sci.*
- Dolev, S., Gilboa, N., and Kopeetsky, M. (2010). Computing multi-party trust privately: in  $O(n)$  time units sending one (possibly large) message at a time. In *SAC'10*. ACM.
- Du, W. and Atallah, M. J. (2001). Privacy-preserving cooperative statistical analysis. In *ACSAC '01*, pages 102–110.
- Du, W. and Zhan, Z. (2002). A practical approach to solve secure multi-party computation problems. In *NSPW'02*. ACM.
- Dumas, J.-G. and Hossayni, H. (2012). Matrix powers algorithm for trust evaluation in PKI architectures. In *STM'12, ESORICS 2012*, LNCS.
- Foley, S. N., Adams, W. M., and O'Sullivan, B. (2010). Aggregating trust using triangular norms in the keynote trust management system. In *STM'2010*.
- Fousse, L., Lafourcade, P., and Alnuaimi, M. (2011). Benaloh's dense probabilistic encryption revisited. In *AFRICACRYPT'11*.
- Goethals, B., Laur, S., Lipmaa, H., and Mielikäinen, T. (2005). On private scalar product computation for privacy-preserving data mining. In *ICISC'04*, LNCS.
- Guha, R. V., Kumar, R., Raghavan, P., and Tomkins, A. (2004). Propagation of trust and distrust. In *WWW'2004*.
- Huang, J. and Nicol, D. M. (2010). A formal-semantics-based calculus of trust. *IEEE Internet Computing*.
- Jøsang, A. (2007). Probabilistic logic under uncertainty. In *CATS'2007*.
- Lafourcade, P. and Puys, M. (2015). Performance evaluations of cryptographic protocols verification tools dealing with algebraic properties. In *FPS'15*.
- Lindell, Y. (2009). Secure computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining, Second Edition 4 Volumes*. IGI Global.
- Michalas, A., Dimitriou, T., Giannetos, T., Komninos, N., and Prasad, N. R. (2012). Vulnerabilities of decentralized additive reputation systems regarding the privacy of individual votes. *Wireless Personal Communications*, 66(3):559–575.
- Mohassel, P. (2011). Efficient and secure delegation of linear algebra. *IACR Cryptology ePrint Archive*.
- Ozarow, L. H. and Wyner, A. D. (1984). Wire-tap channel II. In *EUROCRYPT'84*.
- Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*.
- Shamir, A. (1979). How to share a secret. *CACM*, 22(11).
- Wang, I.-C., Shen, C.-H., Hsu, T.-S., Liao, C.-C., Wang, D.-W., and Zhan, J. (2008). Towards empirical aspects of secure scalar product. In *ISA'08*.
- Yao, A. C. (1982). Protocols for secure computations. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*.