Dual Protocols for Private Multi-party Matrix Multiplication and Trust Computations*

Jean-Guillaume Dumas[1], Pascal Lafourcade[2], Jean-Baptiste Orfila[1] and Maxime Puys[3]

[1]Université Grenoble Alpes, CNRS, LJK, UMR 5224, 700 av. centrale, IMAG/CS-40700, 38058 Grenoble Cedex 9, France.
{Jean-Guillaume.Dumas,Jean-Baptiste.Orfila}@univ-grenoble-alpes.fr
[2]Université Clermont Auvergne, LIMOS, UMR 6158, Campus Universitaire des Cézeaux, BP 86, 63172 Aubière Cedex, France.
pascal.lafourcade@uca.fr
[3]Université Grenoble Alpes, CNRS, Verimag, UMR 5104, 700 av. centrale, IMAG/CS-40700, 38058 Grenoble Cedex 9, France.
Maxime.Puys@univ-grenoble-alpes.fr

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 two dual protocols 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 cryptographic protocol verification tools, we are able to check the security of both protocols and provide a countermeasure for each attack found by the tools. We also give a generic 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 [40] 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

*This work was partially supported by "Digital trust" Chair from the University of Auvergne Foundation, by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025) and by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

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 [8] and many tools exist to realize MPC protocols. They comprise for instance the use of a Trusted Third Party [19], the use of Shamir's secret sharing scheme [36], or more recently the use of homomorphic encryption [25]. It is also possible to mix these techniques [15].

Our goal is to apply MPC to the distributed evaluation of trust, as defined in [28]. Indeed, 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 [26]. 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 [28]. In *e.g.* [23] algorithms are proposed to quantify the trust relationship between two entities in a network, using transitivity and reachability. Then the authors of [27] applied the associated calculus of trust to public key infrastructures. For instance, in [20], 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 [30]. 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 [18, 1, 39]. 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. Then, to reduce some communication costs of sharing schemes, the use of homomorphic encryption is natural [25]. In such cases the underlying homomorphic cryptosystem is quite often that of Paillier [34] or of Benaloh [9, 24].

Now, computing a dot product with n players is actually close to the MPWP protocol of [17], 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. The original MPWP protocol has to use a Benaloh-like homomorphic encryption. We here show how to use instead a Paillier-like system, usually faster in practice. Also, in MPWP the total volume of communication for a dot product is $O\left(n^3\right)$ with $O(n)$ communication rounds. Intuitively, by making the first player masking each term in the dot-product sum, instead of having all the players mask their own, we can get rid of the secret sharing part of MPWP. This enables us first to reduce the amount of communications from

$O(n^3)$ to $O(n)$ and, second, to reduce the number of communication rounds. Indeed instead of having $O(n)$ steps, we can use a setting with $\lfloor \frac{n-1}{2} \rceil$ parallel rounds, each one with less than five parallel communication steps. Thus, we propose in Section 6 a way to reduce the parallel complexity of the protocol (that could also apply to MPWP): we replace a ring of n players into parallel rings of 3 or 4. Security is modified, as the new protocol is secure against semi-honest adversaries while the initial one was secure against a coalition of malicious ones. Now we also propose a generic mitigation scheme, which we call a *random ring order*, that allows to recover the security against malicious adversaries, with high-probability, while preserving efficiency. We then obtain a first secure and efficient protocol.

Another possibility is to use a two-phases protocol sketched in [41]: this $O(n)$ protocol requires to homomorphically exchange vector coefficients in a first phase. Then, it uses a multiparty summation protocol. In [41] it is suggested to use protocols by [5] for the summation in the second phase. We show here that this summation protocol is not resistant against a coalition of malicious insiders. To repair the protocol, one can use instead a secret sharing scheme, but this is back to an $O(n^2)$ communication protocol. We here instead propose first to use Paillier-like homomorphic schemes within the whole protocol. Second we prove that it is possible to use a classical salary-sum protocol for the summation phase, thanks to our novel random ring order mitigation scheme. We thus also preserve both advantages, a $O(n)$ time and communications costs as well as security against malicious adversaries. This resulting protocol is then actually somewhat dual to our first one.

Note that several other generic MPC protocols exist, usually evaluating circuits, but they require $O(n^3)$ computations and/or communications per dot-product [10, 15].

Contributions. Overall, we provide the following results:

1. A fully secure 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 $O(n^2)$.

2. An $O(n)$ time and communications protocol, called *Distributed and Secure Dot-Product* ($DSDP_n$) (for n participants), which allows us to securely compute a dot product UV, against a semi-honest adversary, where one player, the master, owns a vector U and where each player knows one coefficient of V.

3. A parallel variant that performs the dot-product computation in parallel among the players and thus uses a constant total number of rounds. This is extended to a *Parallel Distributed and Secure Matrix-Multiplication* ($PDSMM_i$) family of protocols.

4. A dual novel approach, *YTP-SS*, adapted from a combination of [41] and [5], together with its security analysis, and with the same characteristics:

   - Against malicious adversaries it yields an $O(n^2)$ communications protocol, using homomorphic encryption and secret sharing.
   - Against semi-honest adversaries it yields a novel, $O(n)$ time and communications protocol. Similarly, our *random ring order* mitigation can be applied to it, in order to preserve privacy with high probability, also against a coalition of *malicious insiders*.

Both approaches, *DSDP* and *YTP-SS*, detailed later respectively in Section 4 and Section 7, are sketched in Figure 1:

- in *DSDP*, the master player first receives the encrypted coefficients of V; then, second, he homomorphically multiplies them by his U coefficients while masking everything with random nonces (s_i in Figure 1, left); third, the dot-product is recovered by a distributed homomorphic sum.
- in *YTP-SS*, the master player first sends its encrypted coefficients of U; then, second, the other players homomorphically multiply them by their coefficients of V while masking everything with random nonces (s_i in Figure 1, right); third the sum of nonces is computed in a distributed manner as a homomorphic salary-sum.

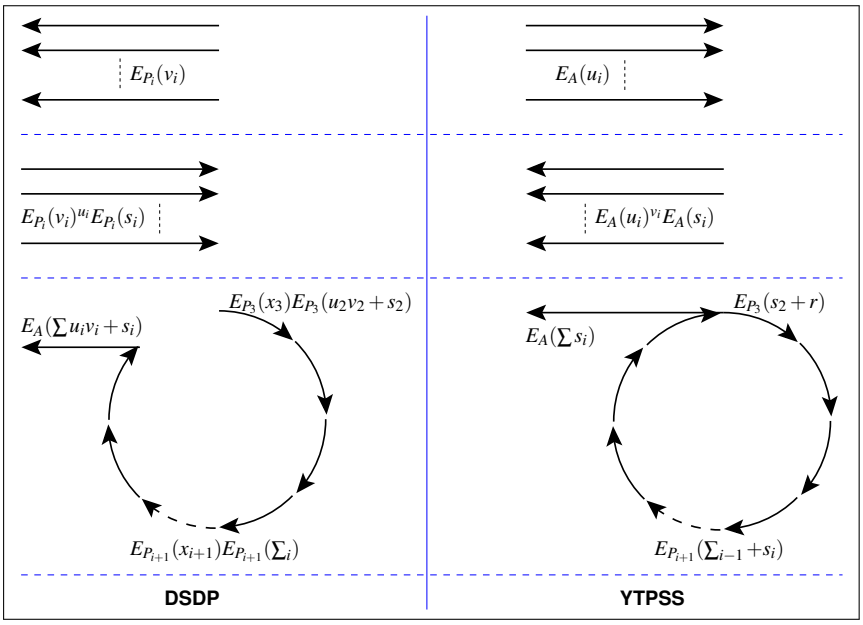

Figure 1: Our dual approaches for secure distributed dot products (left: *DSDP*, see Section 4; and right: *YTP-SS*, see Section 7)

5. We also provide a security analysis of both *DSDP* and *YTP-SS* protocols using ProVerif [12, 11], a cryptographic protocol verification tool. Such analysis 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 [33] and where the players take random order in the protocol to preserve privacy with high probability, even against a coalition of *malicious insiders*.

   We also use AVISPA [3] in order to compare the security of the salary sum protocol [35] with the summation protocol from [5]. More precisely we used OFMC [6] or CL-ATSE [38] the two backend tools of AVISPA suite that are able to deal with several exclusive-or operators. This analysis allows us to safely

replace the summation protocol from [5] used in [41] by the lighter salary sum protocol [35] in order to obtain *YTP-SS*.

6. 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.

A preliminary subset of these results were presented at the Secrypt'16 conference [21]. Here, we first recall these results and then develop a whole new dual approach. We detail this new approach and its associated security analysis and track some attacks with more automatic verification tools. As a side effect we show that the protocol for private sum in [5] is in fact not more secure than a classical salary sum [35], in a malicious setting. Comparing with [21], we also give here a detailed description of the new efficient algorithms: we have now complete formalizations for private sum and private dot-products, as well as effective arbitrary precision implementations in C++ for private multiparty matrix multiplication. Both the formalizations and the implementations are available online in this web-site: matmuldistrib.forge.imag.fr. For instance, we have ran practical performance evaluation on realistic inputs: nowadays, more than 700 different certification authorities are very active [22]; we show that with our new schemes they could efficiently compute their complete trust relationships, and thus enhance the security of public-key infrastructures, by performing a distributed private computation taking less than 10 minutes overall.

Outline. In Section 2, we thus first recall some multi-party computation notions. We then present in Section 3 our quadratic variant of MPWP and a linear-time protocol in Section 4. The associated security proofs and countermeasures are given in Section 5 and we present parallelized versions in Section 6. A dual approach is then presented in Section 7, that now allow for the faster implementation. Finally, in Section 8, we show how our protocols can be adapted to perform a private multi-party, monoid-based, trust computation in a large network.

2 Background

We use a public-key homomorphic encryption scheme where both addition and multiplication are considered. There exist many homomorphic cryptosystems, see for instance [32, § 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 $\mathrm{Add}(c_1;c_2)$, on ciphered messages and one modular multiplication, denoted by $\mathrm{Mul}(c;m)$, between a ciphered message and a cleartext. That is, $\forall m_1,m_2 \in \mathbb{Z}/m\mathbb{Z}$: $\mathrm{Add}(E(m_1);E(m_2)) = E(m_1+m_2 \mod m)$ and $\mathrm{Mul}(E(m_1);m_2) = E(m_1 m_2 \mod m)$. For instance, Paillier's or Benaloh's cryptosystems [34, 9, 24] can satisfy these requirements, *via* multiplication in the ground ring for addition of enciphered messages ($\mathrm{Add}(E(m_1);E(m_2)) = E(m_1)E(m_2) \mod m$), and *via* exponentiation for ciphered multiplication ($\mathrm{Mul}(E(m_1);m_2) = E(m_1)^{m_2} \mod m$), we obtain the following homomorphic properties:

$$E(m_1)E(m_2) = E(m_1+m_2 \mod m) \tag{1}$$

$$E(m_1)^{m_2} = E(m_1 m_2 \mod m) \tag{2}$$

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 From MPWP to P-MPWP

We first describe the *MPWP* protocol [17], and then we explain modifications made to construct *P-MPWP* a lighter version of *MPWP*.

3.1 MPWP Description

The *MPWP* protocol [17] 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.,* $TV = [E_1(w_2), \ldots, E_1(w_n)]$. 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 $Trust = D_1(A) - \sum_{j=2}^{n} D_1(\gamma_j) = D_1(A) - \sum_{j=2}^{n} PSS_j = D_1(A) - \sum_{j=2}^{n} \sum_{i=2}^{n} z_{i,j} = D_1(A) - \sum_{i=2}^{n} z_i = \sum_{i=2}^{n} u_i v_i$.

3.2 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:

$$\forall i, 0 \leq u_i \leq B, 0 \leq v_i \leq B \tag{3}$$

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

$$(n-1)B^2 < M. \tag{4}$$

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 1. *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 $(n-1)B < N_i$, $\forall i = 2, \ldots, n$, 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 [34, 24, 9], 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 [31]. 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*.

4 A Linear Dot Product Protocol

We first provide a protocol for 3 players. We then generalize it for n and give a security proof for a semi-honest intruder.

4.1 Overview with Three Players

We first present in Figure 2 our $DSDP_3$ protocol (Distributed and Secure Dot-Product), for 3 players. This is a detailed version of Figure 1, left. 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. First, Bob and Charlie have to share their private values with Alice. They need to cipher the coefficients v_2 and v_3 to hide them from Alice. Hence, Bob uses his public key to cipher v_2 and obtains $c_2 = E_{pubB}(v_2)$. Charlie follows the same process by using his own public key and gets $c_3 = E_{pubC}(v_3)$. Then, both players send their coefficients, encrypted, to Alice. Next, she homomorphically multiplies each one of these by her u_i coefficients, e.g. $c_2^{u_2} = E_{pubB}(v_2 u_2)$ and $c_3^{u_3} = E_{pubC}(v_3 u_3)$. Subsequently, she picks a random value r_2, and ciphers it using the public key of Bob. Alice homomorphically multiplies the

obtained $E_{pubB}(r_2)$ with the previous ciphered product (i.e. $c_2^{u_2} = E_{pubB}(v_2u_2)$) and gets $\alpha_2 = E_{pubB}(v_2u_2 + r_2)$. Hence, she masked her private coefficient u_2 using the random value r_2. Similarly, she computes $\alpha_3 = E_{pubC}(v_3u_3 + r_3)$ with another random value r_3 and the public key of Charlie. Finally, she sends both values to Bob.

As the owner of associated private key, Bob deciphers α_2 and obtains $\Delta_2 = v_2u_2 + r_2$. Then, he ciphers Δ_2 using Charlie's public key. He is now able to homomorphically multiply the latter cipher with α_3. The result, denoted $\beta_3 = E_{pubC}(v_3u_3 + r_3 + v_2u_2 + r_2)$, is sent to Charlie. Charlie starts by deciphering β_3 to get $\Delta_3 = v_3u_3 + r_3 + v_2u_2 + r_2$. Next, he ciphers the previous value with the public key of Alice, sends her the obtained $\gamma = E_{pubA}(v_3u_3 + r_3 + v_2u_2 + r_2)$. The security of the protocol is based on the use of homomorphic encryption schemes and random values. The formal proof is detailed in Section 5. However, based on the previous description and Figure 2, the intuition is the following:

- At the beginning, Alice only received ciphered values, so that she cannot deduce any information ;
- During the next phase, the other players decrypt intermediate values (α_i, β_i). They obtain equations containing at least three terms $u_iv_i + r_i$: 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, v_2 and v_3, but a single equation.

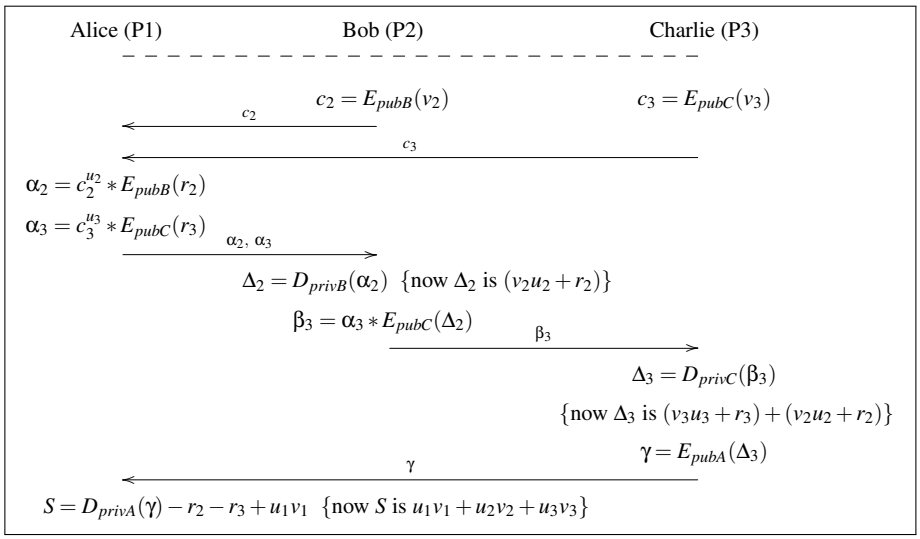

Figure 2: $DSDP_3$: Secure dot product of vectors of size 3 with a Paillier-like asymmetric cipher.

After several decryptions and re-encryptions, and removal of the random values r_i, we obtain $S = \sum u_iv_i$. The homomorphic Properties (1) and (2) only guaranty that $D(\text{Add}(\text{Mul}(E(v_i); u_i); r_i)) = v_iu_i + r_i \mod N_i$, 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 $S_4 = ((u_2v_2 + r_2) \mod N_2 + u_3v_3 + r_3) \mod N_3$. She can remove r_3, *via* $S_3 = S_4 - r_3 \mod N_3$, but then $S_3 = ((u_2v_2 + r_2) \mod N_2 + u_3v_3) \mod N_3$. 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: $S_3 = (u_2v_2 + r_2) \mod N_2 + u_3v_3$. Then Alice can remove the second random value, this time modulo N_2: $S_2 = (u_2v_2 + u_3v_3) \mod N_2$, where now $N_2 > 2B^2$ suffices to recover $S = S_2 \in \mathbb{N}$. We generalize to n players this protocol in the following section.

4.2 General Protocol with n Players

We give the generalization $DSDP_n$, of the protocol of Figure 2 for n players in Algorithm 2 hereafter. For this protocol to be correct, we use the previously defined bound (3) on the players' private inputs.

Algorithm 2 $DSDP_n$ 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 $S = U^T V$.

for $i = 2 \dots n$ **do**
  $P_i : c_i = E_i(v_i)$
  $P_i \xrightarrow{c_i} P_1$
for $i = 2 \dots n$ **do**
  $P_1 : r_i \xleftarrow{\$} \mathbb{Z}/N_i\mathbb{Z}$
  $P_1 : \alpha_i = c_i^{u_i} * E_i(r_i)$ so that $\alpha_i = E_i(u_iv_i + r_i)$
$P_1 \xrightarrow{\alpha_2} P_2$
for $i = 2 \dots n-1$ **do** $P_1 : \xrightarrow{\alpha_{i+1}} P_i$
$P_2 : \Delta_2 = D_2(\alpha_2)$ so that $\Delta_2 = u_2v_2 + r_2$
$P_2 : \beta_3 = \alpha_3 * E_3(\Delta_2)$ so that $\beta_3 = E_3(u_3v_3 + r_3 + \Delta_2)$; $P_2 \xrightarrow{\beta_3} P_3$
for $i = 3 \dots n-1$ **do**
  $P_i : \Delta_i = D_i(\beta_i)$ so that $\Delta_i = \sum_{k=2}^{i} u_kv_k + r_k$
  $P_i : \beta_{i+1} = \alpha_{i+1} * E_{i+1}(\Delta_i)$ so that $\beta_{i+1} = E_{i+1}(u_{i+1}v_{i+1} + r_{i+1} + \Delta_i)$; $P_i \xrightarrow{\beta_{i+1}} P_{i+1}$
$P_n : \Delta_n = D_n(\beta_n)$; $P_n : \gamma = E_1(\Delta_n)$; $P_n \xrightarrow{\gamma} P_1$
return $P_1 : S = D_1(\gamma) - \sum_{i=1}^{n-1} r_i + u_1v_1$.

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 3.

Lemma 3. *Under the bound (3), and for any r_i, let $M_2 = (u_2v_2 + r_2) \mod N_2$ and $M_i = (M_{i-1} + u_iv_i + r_i) \mod N_i$, for $i = 2 \dots n-1$. Let also $S_{n+1} = M_n$ and $S_i = (S_{i+1} - r_i)$*

$\bmod N_i$ *for* $i = n \ldots 2$. *If we have:*

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

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 $i = n..3$: indeed $S_n = (M_n - r_n) \mod N_n = (M_{n-1} + u_n v_n) \mod N_n$. 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 $3 \leq i < n$, $S_i = (S_{i+1} - r_i) \mod N_i = (M_i + \sum_{j=i+1}^{n} u_j v_j - r_i) \mod N_i = (M_{i-1} + u_i v_i + r_i + \sum_{j=i+1}^{n} u_j v_j - r_i) \mod N_i = (M_{i-1} + \sum_{j=i}^{n} u_j v_j) \mod N_i$, 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, $S_2 = (S_3 - r_2) \mod N_2 = (M_2 + \sum_{j=3}^{n} u_j v_j - r_2) \mod N_2 = (\sum_{j=2}^{n} u_j v_j) \mod N_2$. As $\sum_{j=2}^{n} u_j v_j < (n-1)B^2$, by (5) for $i = 2$, we have $S_2 = \sum_{j=2}^{n} u_j v_j \in \mathbb{N}$. □

This shows that the $DSDP_n$ protocol of Algorithm 2 can be implemented with a Paillier-like underlying cryptosystem, provided that the successive players have increasing modulo for their public keys.

Theorem 4. *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 2 is correct. It requires* $O(n)$ *communications and* $O(n)$ *encryption and decryption operations.*

Proof. First, each player sends his ciphered entry 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 Δ_i. By induction, we obtain $\Delta_i = \sum_{k=2}^{i} u_k v_k + r_k$. This value is then re-enciphered 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 α_i; $n-1$ decryptions for the Δ_i; $n-1$ encryptions, homomorphic operations and communications for the β_i; and finally one encryption and one communication for γ. Then P_1 needs $O(n)$ operations to recover S. □

5 Security of the $DSDP_n$ 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 [12], an automatic verification tool for cryptographic protocols.

5.1 Security Proofs

The standard security definition in MPC models [30] covers actually many security issues, such as correctness, inputs independence, privacy, etc. We first prove that under these settings, computation of the dot product is safe.

Lemma 5. *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. □

Then, proving the security relies on a comparison between a real-world protocol execution and an ideal one. The latter involves a 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 6. *In the presence of a set C of semi-honest adversaries with inputs set X_C, a protocol Π 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 $\{C, \{View^{\Pi}_{P_i}\}_{P_i \in C}\}$.*

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

Lemma 7. *By assuming the semantic security of the cryptosystem E, for $n \geq 3$, $DSPD_n$ 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 γ' value is computed using R along with the protocol output S: $\gamma' = E_1(S + \sum_i^{n-2} r_i + u_1 v_1)$. 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 γ instead of β_{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', γ', C' (except c_i) is made by ciphering random values using the adequate public key. More precisely, c_i is ciphered using v_i

and the public key of P_i. For Δ'_i, the simulator Sim_i has to forward the random value previously chosen to be ciphered as α_i. Indistinguishability is based on the semantic security of E (for A, B, C and γ) and on the randomness added by P_1 (and thus unknown by P_i). Then, Δ'_i is computationally indistinguishable from the real Δ_i. Hence, $View_{P_i}$ and $View_{S_i}$ are indistinguishable and $DSDP_n$ is secure against one semi-honest adversary. □

5.2 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 [12, 11]. 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 [16]. Moreover, as showed in [29], some equational theories such as Exclusive-Or can already outspace 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 α_i messages that Bob receives as follows:

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

This property allows Bob to obtain $u_2v_2+r_2$ from α_2. This also allows an intruder to simulate such calculus and impersonate Bob. We also model:

(ii). β_3 by $\forall u,v,r,x,y,z,k,\ charlie(E_k(uv+r),E_k(xy+z)) = E_k(uv+xy+r+z)$

(iii). β_4 by $\forall u,v,r,x,y,z,a,b,c,k,\ dave(E_k(uv+xy+r+z),E_k(ab+c)) = E_k(uv+xy+ab+r+z+c)$

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 and we reach the limits of ProVerif (it does not terminate for the corrected version of the protocol). In this situation it is possible to guide the analysis of ProVerif through the *nounif* keyword. It basically says to ProVerif to cut specified paths. It can be usefull to remove possible loops or just to speed up the analysis in some cases. However, it is far from being trivial because it may remove the completeness of the analysis, doing such kind of trick requires to be sure that the cut paths do not lead to attacks which is difficult. It is better to leave the default mode of the tool for this. All the associated source files are available in a web-site: matmuldistrib.forge.imag.fr.

Analysis in case of a passive adversary. Using these equational theories on the protocol described in Figure 2, 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 learn by the intruder for any combinations of leaked key. This confirms the proofs given in Section 5.1 against the *semi-honest* adversaries.

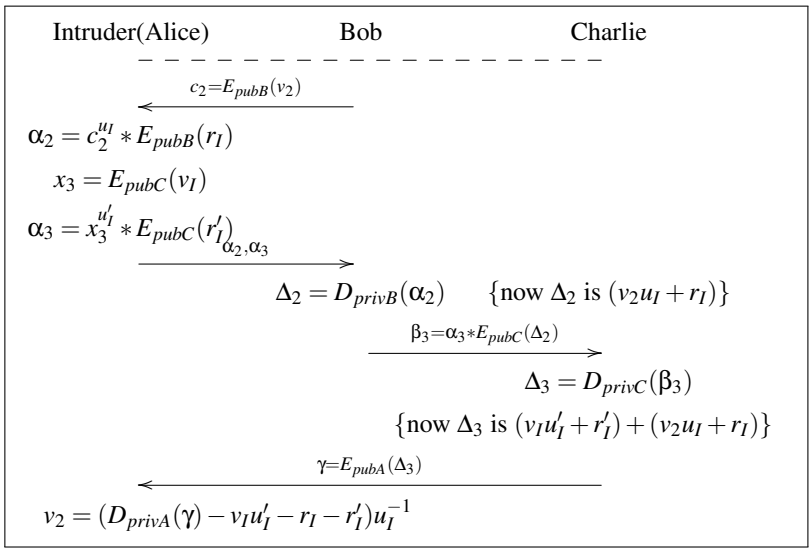

Figure 3: Attack on the secrecy of v_2

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.

(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 6.2, 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, generated by Bob, is then presented in Figure 3.

Bob sends his secret value v_2, encrypted as c_2 in Figure 3 to Alice. The malicious adversary usurps Alice and générate fakes values r_i and v_i for all participants. Those values are used in the protocol as α_i instead of the one generated by Charlies, Dave, etc. Thus v_2 is the only value unknown to the adversary. The protocol continues normaly (Δ_2 is sent to Bob who computes Δ_3 and sends it to Charlie, etc.). At the end of the protocol, the adversary obtains the computed result γ. As said earlier, γ is made the values known from the adversary except v_2. Thus the adversary 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 α_i except α_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 $u_2v_2 + r_2 + \sum_{i=3}^{n} x_i$, 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 α_i can be counteracted by zero-knowledge proofs of knowledge. P_1 has to prove to the other

players that α_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 [13] given in Figure 4.

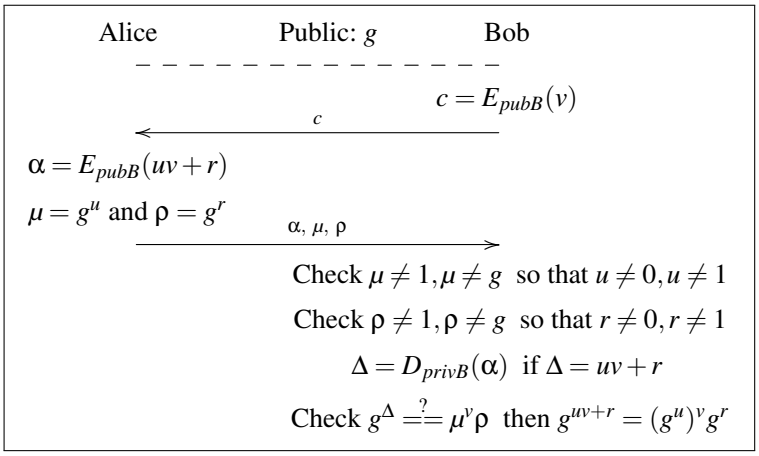

Figure 4: Proof of a non trivial affine transform

In the Protocol 2, this proof of a non trivial affine transform applies as is to α_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 12 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 δ_{i-1}, the check of player P_i ends with $\delta_i = g^{\Delta_i} \stackrel{?}{=} \mu_i^{v_i}\rho_i\delta_{i-1}$. 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 α_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 β_i growing to (β_i, δ_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 α_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 $\Delta_3 = (v_2u_2 + r_2) + (v_3u_3 + r_3)$ 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 $\Delta_{i-1} = (u_2v_2 + ... + u_{i-1}v_{i-1} + r_2 + ... + r_{i-1})$.

(ii). P_{i+1} knows $\Delta_{i+1} = (u_2v_2 + ... + u_{i-1}v_{i-1} + u_iv_i + u_{i+1}v_{i+1} + r_2 + ... + r_{i-1} + r_i + r_{i+1})$.

Thus, by simplifying Δ_{i-1} and Δ_{i+1}, the malicious adversary can obtain $u_i v_i + u_{i+1} v_{i+1} + r_i + r_{i+1}$ 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 6.2 that these kinds of threats can be diminished if the protocol is replayed several times in random orders.

6 Parallel Approach

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 $s = u_1 v_1 + u_2 v_2 + u_3 v_3$. 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 $u_1 v_1$ only after all the computations. For this it is sufficient to modify line 15 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 8.

Algorithm 8 $ESDP_n$ 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 $DSDP_{n+1}(P_1 \dots P_{n+1}, [0, U], [0, V])$.

6.1 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 9. 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.

Theorem 10. *The $PDSMM_n$ Protocol in Algorithm 9 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 9 gives the coefficient c_{ij} such that $c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj}$. First, the

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

k_i coefficients are just the values $1 \ldots (i-1)$ and $(i+1) \ldots n$ 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 2, requires 4 rounds with a constant number of operations: one round for the c_i, one round for the α_i, one round for β_3 and one round for γ. As shown in Algorithm 2, $ESDP_3$, like $DSDP_4$, requires only a single additional round for β_4. □

We have proven the security for one execution of the protocol. Using the transformation of Arapinis et al. [2], we can generalize it to several execution. This transformation only requires that all participants (honest and malicious ones) share a nonce and their identity at the beginning of the protocol. Then these extra information is added into each messages passed to all cryptographic primitives used in the protocol. Then we obtain a version of the protocol secure for an unbounded number of sessions. This approach allows us to have a secure protocol for any number of session without changing the spirit of our proposal.

6.2 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. A similar proposition is sketched

to work with an honest majority in [14, § 2.1]. We here extend it for any number of malicious adversaries up to $n-2$ and analyze the expected behavior. A similar idea, with disjoint orders, is presented also in [37] but with $O(n)$ deterministic repetitions. We will show next than on average much less repetitions are needed. The idea is that to access a given private value, the malicious adversaries have to be well placed in *every* repetition 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. The fundamental property behind our random ring order is that even if we have some malicious colluding participants, the protocol should be safe because the order is chosen uniformly at random over all existing orders. Then the probability that the intruders are always between the same attacked participant is very small. Hence in practice we only need that the random ring order is publicly known by every participant and that it is random. For instance we can use a cryptographic hash function that takes in input the public keys of all participants and the start time of the protocol, that is known by all participants (by publishing it on a website for instance or broadcasting it at the beginning). It allows us to have a unique random ring order per session of the protocol.

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 [33], as presented in Algorithm 11.

Algorithm 11 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 12. *Algorithm 11 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 $S_1 = \sum_{j=2}^{n} u_j \left(v_j - \sum_{i=2}^{d} \lambda_{j,i}\right)$. 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 $S_1 = \sum_{j=2}^{n} u_j \left(v_j + \sum_{i=2}^{d}(B - \lambda_{j,i})\right)$. Then in the following occurrences, Alice gets $S_i = \sum_{j=2}^{n} u_j \lambda_{j,i}$. Finally she computes $\sum_{i=1}^{d} S_i - (d-1)B(\sum_{j=2}^{n} u_j) = \sum_{j=2}^{n} u_j (v_j + (d-1)B) - (d-1)Bu_j = \sum_{j=2}^{n} u_j v_j$. □

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 13. *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 11 with $d \leq 2\ln\left(\min\{k-1, n-k, \frac{n-1}{2}\}\right)\left(1+\frac{k-1}{n-k-1}\right)$ 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 $T_{n,k}(a) \leq \mathbb{E}_{n,k}(a) + T_{n,k}(a-1) \leq \sum_{i=a}^{3} E_{n,k}(i) + T_{n,k}(2) = (\sum_{i=a}^{3}\frac{1}{i})\frac{2(n-2)}{n-k-1} + T_{n,k}(2)$. With 2 attacked and c saved, we consider $T_{n,k=n-c-2}(2) = \frac{n-2}{c+1}$ so that $T_{n,k}(a) \leq (H_a - \frac{3}{2})\frac{2(n-2)}{n-k-1} + \frac{n-2}{n-k-1}$. Now bounds on the Harmonic numbers give $H_a \leq \ln a$ (see, *e.g.*, [7]) and since $a \leq k-1$ and $a \leq n-k$, this shows also that $2a \leq n-1$. Therefore, $T_{n,k}(a) \leq 2\ln\left(\min\{k-1, n-k, \frac{n-1}{2}\}\right)\frac{n-2}{n-k-1}$. □

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 ε, 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 $n\ln(1/\varepsilon)$:

Proposition 14. *With $n = 2t+1$, the number d of random ring repetitions of Algorithm 11 to make the probability of breaking the protocol lower than ε satisfies $d < n\ln(1/\varepsilon)$ 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 $d \geq n\left(\ln\left(\varepsilon^{-1}\right)\right)$, then $d > (n-1)(-\ln\varepsilon) > \frac{\ln\varepsilon}{\ln\left(1-\frac{1}{n-1}\right)}$, which shows that $(1-\frac{1}{n-1})^d < \varepsilon$. □

Note that using $d = (n-1)/2$ disjoint Hamiltonian paths instead of random ones can be an alternative in this worst case [37].

6.3 Complexities and Security Parameter

Overall, the wiretap variant of Algorithm 11 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 Figure 5, for the average (Theorem 13) and worst (Proposition 14) cases of Algorithm 11, and where the protocols of the previous sections are also compared.

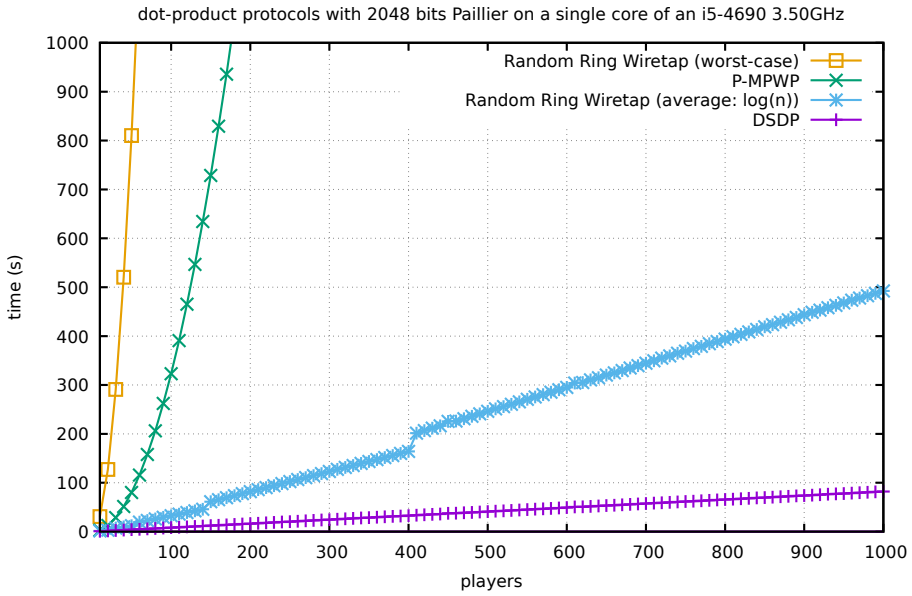

Figure 5: Quadratic and linear protocols timings

On the one hand, we see in Figure 5 that quadratic protocols, with homomorphic encryption, are not usable for a realistic large group of players. On the other 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.

7 *YTP-SS*: A Dual Protocol

Yao, Tamassia and Proctor present in [41] an alternative secure quadratic dot-product protocol, which will be abbreviated as YTP in the following, with a similar data distribution between players. We first explain the idea of this protocol, then we see some attacks on it and finally we propose a new secure linear protocol inspired from this one.

7.1 The Idea of the YTP Protocol

During the first part of the protocol, the master player P_1 (i.e. the one owning a complete vector U) starts by self-ciphering each values u_i for $i \in [2,n]$, and then distributes $E_1(u_i)$ to P_i. Then, using a cryptosystem E verifying the homomorphic properties 1 and 2, each player P_i uses the public key of P_1 and computes $E_1(u_i)^{v_i}E_1(-s_i)$, where

s_i is a random number and v_i the private input. During the next step, each P_i sends the result back to P_1. She computes the product of all received ciphertexts, to finally obtain $D_1(\prod_{i=2}^{n} E_1(u_i v_i - s_i)) = \sum_{i=2}^{n} u_i v_i - s_i$.

In the second phase of the protocol, each P_i will compute the sum of s_i using a secure summation protocol. The authors propose to use an $O(n^2)$ secret sharing scheme or, as a perspective, an alternative sketched in [4, § 5.1]. There, the idea is to compute a secure sum of the randomness between players, and then send the result to the master player. In this protocol, previous random numbers s_i added by each player P_i are hidden with another random number r_i. Then, the players with an even index send to the ones with odd index the $r_i + s_i$ value, while the odd players send only s_i to the even ones. Then, the even (respectively odd) players compute $\sum s_{2i+1}$ (resp. $\sum r_{2i} + s_{2i}$). Further exchanges, bipartite and tree based between players, allow to recover the complete sum. Finally, P_1 is able to recover $\sum_{i=2}^{n} s_i$ from one of the P_i's, and then compute the scalar product.

First, we analyze this approach in the following Section 7.2 and show that it is in fact not more secure than a classical salary sum protocol [35]: a player (e.g. P_2) chooses a random number s and adds it to its private value r_2. Then, it sends to the next indexed player the cipher of $r_2 + s$, using the public key of P_3 i.e. $E_3(r_2 + s)$. Then after deciphering, each player adds their private input to the previous sum and sends the new value by ciphering it using the next player's public key. The last player sends the result to the first one, which simply remove the random value to obtain the sum.

Second we prove that it is possible to use our novel random ring order mitigation scheme with the latter sum. We thus also preserve both advantages, a $O(n)$ time and communications costs as well as security against malicious adversaries. This resulting protocol, *YTP-SS*, is then actually somewhat dual to *DSDP*, as will be shown in Section 7.3.

7.2 Attacks on *YTP* Summation Protocol

In Section 5, we provided automated verification of $DSDP_i$ protocols using ProVerif. We now aim to provide similar analysis for the summation protocol from [4] and compare it to the classical salary sum protocol from [35] to clarify if the exchanges added by the first make it more secure than the second. However, we face the exact same modeling challenge than earlier, that is, modeling of equational theories over abelian groups has been proven undecidable and we have to restrict our analysis to make it feasible. Rather than providing our own hardcoded equational theories as for ProVerif, we choose to model sums as Exclusive-OR operators which is a classic modeling practice. Exclusive-OR is indeed addition modulo two. Moreover, Exclusive-OR has the following four properties: commutativity, associativity, unity, nilpotency. The main difference is that an element is also its inverse, which might introduce some false attacks. Nethertheless, Exclusive-OR is handled by some protocol verification tools and is a reasonable abstraction of addition.

As shown in [29], it appears that ProVerif is barely able to handle Exclusive-OR. Thus we use a different tool named AVISPA [3] to verify *YTP* protocols. AVISPA is a common frontend to four backends analyzers sharing the same input language. This allows us to test modelings against the two backends that are able to handle Exclusive-OR operators, namely Cl-Atse [38] and OFMC [6]. Again, all the associated source files are available in the web-site: matmuldistrib.forge.imag.fr.

Against the salary sum protocol from [35], we check the secrecy of the r_i terms. Both Cl-Atse and OFMC do not find any attack with zero or one corrupted participants. With two corrupted participants both tools find the exact same attacks as in *DSDP* protocols when the attackers are situated before and after an honest participant i. By computing the difference of their outputs they can learn the secret value of the honest participant: $(s+\sum_{j=2}^{i+1} r_j)-(s+\sum_{j=2}^{i-1} r_j) = r_i + r_{i+1}$ (with r_{i+1} known by one corrupted player).

Against the protocol from [4], we check the secrecy of the s_i terms that players aim to sum. No attack is found in absence of corrupted participant while with one, Cl-Atse find two attacks. However after careful review, it appears that both attacks are present due to the approximation of the sum into Exclusive-OR operators. The author of the tool have been notified about this behavior and confirmed that current version of the tool cannot find these attacks. Thus if there is no attack with one corrupted participant, then the protocol from [4] has the same security than the one from [35].

Interestingly, with two corrupted participants, we find an attack on the secrecy of all s_i terms, but with the two corrupted participants being in different positions for each attack and not necessarily before and after as for the salary sum. For instance with four players (P_2 to P_5) computing the sum for the master player P_1, if P_2 and P_5 are colluding then they can obtain the secret term s_3 from P_3 with the attack described in Figure 6. According to the protocol, P_2 sends r_2+s_2, P_3 answers with his random value r_3 and forwards the sum $r_2+s_2+r_3+s_3$. Thus together, P_2 and P_5 know $r_2+s_2+r_3+s_3$, r_2, s_2, r_3 and they can obtain s_3.

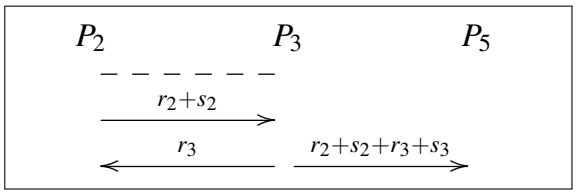

Figure 6: Attack on s_3 in the sum protocol from [4]

7.3 From *YTP* to *YTP-SS*: a Dual Protocol to *DSDP*

From a global point of view, *DSDP* and *YTP* protocols follow the same pattern: a first step of inputs hiding and a second one of randomnesses removing (sketched in Figure 1). However, we remark that strategies employed during the first phase are somewhat dual. Indeed, in *DSDP*, the method is *all-for-one* whereas a *one-for-all* approach is used in *YTP*. The second phases are quite different, *DSDP* uses of a ring topology while *YTP* uses a tree-based bipartite one. Nevertheless, none of the protocols offers security against malicious adversaries nor collusion attacks. Our idea is then to apply the *Random Ring Order* (*RRO*) mitigation defined in Section 6.2 to obtain an *a priori* threshold determining a tradeoff between performance and security.

However, instead of just applying the *RRO*, we start by increasing the overall efficiency of the *YTP* protocol. Indeed, we have shown in the previous Section 7.2 that the summation protocol of [4] does not actually improve the security with regards to simpler summation protocols. We therefore propose to use the simpler ones, in order to obtain a linear communication cost in the number of players, and therefore a linear number of encryptions during the summation phase. The obtained variant is denoted

YTP-SS. An example of the protocol execution with three players is illustrated in Figure 7 (detailed version of Figure 1, right), and the complete protocol is described in Algorithm 15.

Algorithm 15 *YTP*-SS protocol

Require: $n \geq 3$ players, two vectors U and V such that P_1 knows complete vector U, and each players P_i knows the component v_i of V, for $i = 1 \dots n$;
Require: E_i (resp. D_i), encryption (resp. decryption) function of P_i, for $i = 1 \dots n$.
Ensure: P_1 knows the dot-product $S = U^T V$.

1: **for** $i = 2 \dots n$ **do**
2: $P_1 : c_i = E_1(u_i)$
3: $P_1 \xrightarrow{c_i} P_i$
4: $P_i : r_i \xleftarrow{\$} \mathbb{Z}/N_1\mathbb{Z}$
5: $P_i : \alpha_i = c_i^{u_i} * E_1(-r_i)$ so that $\alpha_i = E_1(u_i v_i - r_i)$
6: $P_i : \xrightarrow{\alpha_i} P_1$
7: $P_1 : S_a = \sum_{i=2}^{n} D_1(\alpha_i)$ so that $S_a = \sum_{i=2}^{n} u_i v_i - r_i$
8: $P_2 : s \xleftarrow{\$} \mathbb{Z}/N_1\mathbb{Z}$
9: $P_2 : \beta_3 = E_3(r_2 + s)$
10: $P_2 : \xrightarrow{\beta_3} P_3$
11: **for** $i = 3 \dots n-1$ **do**
12: $P_i : \Delta_i = D_i(\beta_i)$
13: $P_i : \beta_{i+1} = E_{i+1}(\Delta_i + r_i)$
14: $P_i : \xrightarrow{\beta_{i+1}} P_{i+1}$
15: $P_n : \Delta_n = D_n(\beta_n)$
16: $P_n : \beta_2 = E_2(\Delta_n + r_n)$
17: $P_n : \xrightarrow{\beta_2} P_2$
18: $P_2 : S_b = D_2(\beta_2) - s$ so that $S_b = \sum_{i=2}^{n} r_i$
19: $P_2 : \xrightarrow{E_1(S_b)} P_1$
20: **return** $P_1 : S = S_A + D_1(E_1(S_b)) + u_1 v_1 = \sum_{i=1}^{n} u_i v_i$.

As in Section 5.1, we have the following Lemma 16 for the security of the protocol.

Lemma 16. *By assuming the semantic security of the cryptosystem E, for $n \geq 3$, $YTPSS_n$ is secure against one semi-honest adversary.*

Proof. We suppose that the partial homomorphic cryptosystem E is semantically secure. We show next that the views of players in real and ideal executions can only be distinguished with a negligible probability. The protocol requires three different player's roles: the master player (owning the complete U vector), a sub-master player (determining the random s used for computing S_b), and the remaining players. Therefore, the proof is cut in three parts, one for each possibility of corruption. In each part, we describe the algorithm of the simulator Sim_A (where A denotes the corrupted player). This simulator generates the view of the protocol for player A, using as inputs and random numbers the ones from A, and using as the output of the protocol what is obtained from the TTP (Trusted Third Party) in the ideal world. We recall that protocols inputs are U for P_1, v_i for P_i ($i \in \{2..n\}$), and output is $S = \sum_{i=2}^{n} u_i v_i$ for P_1 and nothing for the others players. Apart from the notations used in the Protocol 15, we

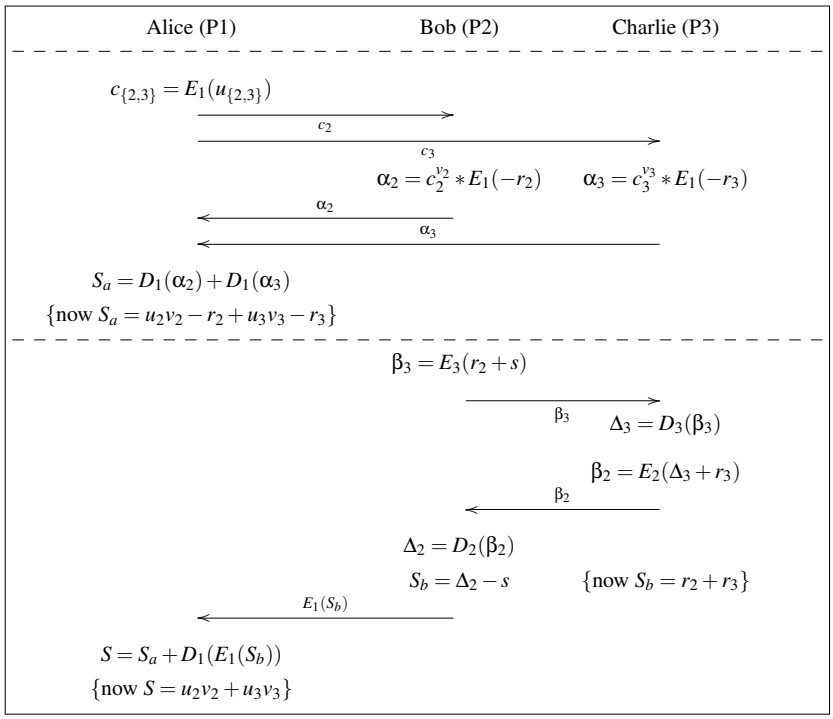

Figure 7: *YTP-SS* protocol to securely compute dot-product using homomorphic encryption scheme E

define also: $\gamma = E_1(S_b)$, $U = \{u_i\}_{2\leq i\leq n}$, $A = \{\alpha_i\}_{2\leq i\leq n}$, $B = \{\beta_i\}_{2\leq i\leq n}$, $C = \{c_i\}_{2\leq i\leq n}$. For a value X obtained from a real execution, we denote by X' the simulated one.

P_1 **is corrupted:** During a protocol execution, P_1 will obtain the following view: $View_{P_1} = \{U, C, A, B, S_a, S_b, S\}$, from her inputs, output and leaked values. Using the U vectors as input, the simulator Sim_{P_1} provides $View_{Sim_{P_1}} = \{U, C', A', B', S'_a, S'_b, S\}$. First, it computes $\forall i \in \{2..n\} : c'_i = E_1(u_i)$. Then, by sampling $n-1$ random numbers w_i, Sim_{P_1} computes $\alpha'_i = E_1(w_i)$ and deduces $S'_a = \sum_{i=2}^{n}(w_i)$. B' values are obtained by ciphering new random numbers z_i using the public key of the i^{th} *players* i.e. $\beta'_i = E_i(z_i)$. Finally, from the output S of the TTP, Sim_{P_1} returns $S'_b = S - S'_a$. Therefore, we have $View_{Sim_{P_1}} = \{U, C', A', B', S'_a, S'_b, S\}$. Both the semantic security of E and the addition of random numbers thus imply the indistinguishability of $View_{P_1}$ and $View_{Sim_{P_1}}$.

$P_i, i \in \{3..n\}$ **is corrupted:** A real protocol execution gives to P_i the view $View_{P_i} = \{v_i, r_i, C, A, B, \Delta_i, \gamma\}$. Knowing v_i and r_i, we describe the algorithm executed by Sim_{P_i} to generate $View_{Sim_{P_i}} = \{v_i, r_i, C', A', B', \Delta'_i, \gamma'\}$. To simulate C', the simulator picks $n-1$ random numbers z_i and ciphers them using the public key of P_i. All the $n-1$ values of B but β_{i+1} are simulated using new random values w_i ciphered with the P_i's key. Then, Δ'_i is equal to w_i, and $\beta'_{i+1} = E_{i+1}(w_i + r_i)$ if $i \leq n-1$ or $\beta'_2 = E_2(w_n + r_n)$ otherwise. For all α'_j values, with $j \neq i$, Sim_{P_i} ciphers again random values t_i. α'_i is obtained by computing: $\alpha'_i = c_i^{v_i} E_1(-r_i)$, since the simulator has access to v_i and r_i. For the γ' value, Sim_{P_i} ciphers another random value with the public key of P_1. By applying the previous arguments, $View_{P_i}$ is indistinguishable from $View_{Sim_{P_i}}$.

P_2 is corrupted: Finally, we describe the algorithm of Sim_2 providing $View_{Sim_{P_2}} = \{v_2, r_2, s, C', A', B', \Delta_2', S_b', \gamma'\}$, where v_2 is the input of P_2 and r_2 as well as s are random numbers chosen by P_2. The only difference between P_i (with $i > 2$) and P_2 is during the summation round, where P_2 provides the random number s and learns S_b from the protocol execution. Then, we use the same construction as for the previous simulator $Sim_{P_i}, i \in \{3..n\}$ to provide a partial view. In order to simulate the S_b' value, the simulator computes $S_b' = \Delta_2' - s$, using the Δ_2' obtained in the partial view. Finally, γ' is equal to $E_1(S_b')$. As previously, indistinguishability is obtained from the addition of random numbers and the semantic security of E.

Note that, in the case where $n = 3$, P_2 is able to deduce the third player's random value from a real protocol execution by computing $r_3 = \Delta_2 - s$. In the ideal case, adversary gets $r_3' = \Delta_2' - s$. Since r_3 is a random value, P_2 cannot make the distinction with r_3'. □

Remark 17. *In the simulation paradigm, the sub-protocol of summation is secure. However, in the case where only two players are involved, P_1 is able to retrieve the secret value r_2 of the other player: therefore the protocol is not safe. Nevertheless, in $YTP\text{-}SS_3$ the summation phase is only applied on random values, so that the knowledge of r_3 by P_2 still preserves the safety property.*

Now for attacks, we used again automated verifications alongside with the above handwritten proofs. As for *DSDP*, we use the ProVerif tool to handle homomorphic equational theories. Surprisingly it appears that *YTP-SS* seems more secure than *DSDP*, when used without random ring order mitigation. In both protocols, ProVerif finds an attack when both Alice and two other participants P_{i-1} and P_{i+1} are corrupted. Then they can obtain the v_i secret from P_i. In the case of *DSDP*, ProVerif was able to find this attack with only Alice (P_1) and Charlie ($P3$). Together they were able to retrieve v_2 from P_2 since Alice is also the player right before P_2 as presented at the end of Section 5.2. However, according to ProVerif, this attack is not possible on *YTP-SS*. This can be intuitively explained since Charlie would obtain $r + s$ instead of r alone from Bob when performing the salary sum. Thus we can conclude according to our experiments with ProVerif that *YTP-SS* provides more security than *DSDP*. Now, when used in parallel, that is in groups of 3 players, the security is identical.

In terms of efficiency, *YTP-SS* and *DSDP* are both linear, with a slight advantage for *YTP-SS*: in the second phase, namely the salary sum, a non-homomorphic encryption can be used. Figure 8 shows that this is indeed the case with Paillier encryption for the whole *DSDP*, whereas Paillier encryption is used in the first phase of *YTP-SS* and plain RSA-OAEP in the second phase.

Using this new version of *YTP-SS*, we can apply the *Random Ring Order* mitigation scheme of Section 6.2, in the exact same way as in *DSDP*. The average number of runs required to prevent malicious attacks is still given by Theorem 13. Indeed, in a parallel setting, we just shown in Section 7.2 that any honest player is also safe if he can be placed with a single other honest player, in at least one of the random ring runs. In consequence, we obtain the same security trade-offs as in Section 6.2.

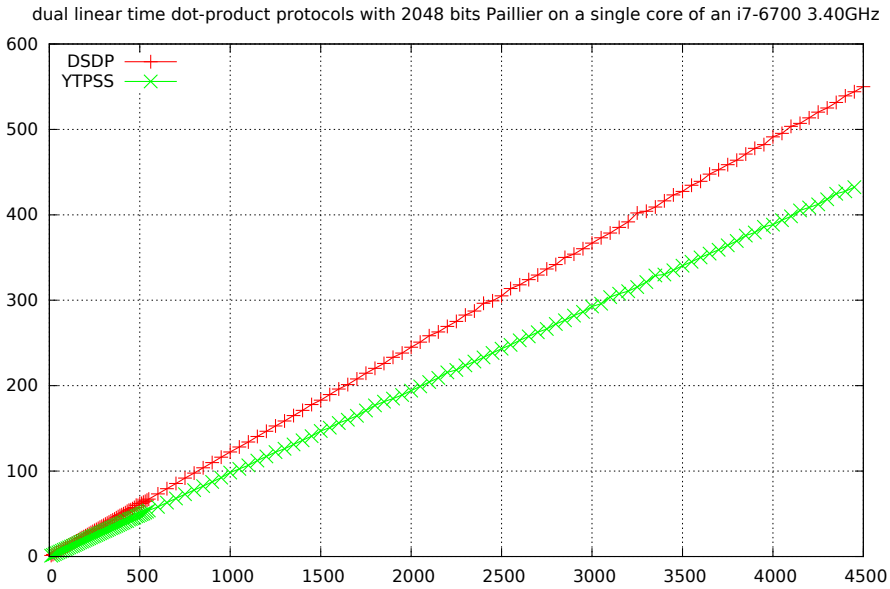

Figure 8: Time executions of *DSDP* and its dual *YTP-SS*

8 Secure Multiparty Computation of Trust in Public-key Infrastructures

We now come back to the aggregation of trust. As sketched in introduction, the first step is to reduce the computation of trust to that of powers of the adjacency matrix. The latter can then be performed by parallel dot-products.

We first recall next, in Section 8.1, the formal definitions of the sequential and parallel aggregations of trust. Then we show, in Section 8.2, that powers of this adjacency matrix can be evaluated privately in a distributed manner, provided than one disposes of an homomorphic cryptosystem satisfying the Properties (1) and (2). Eventually, in Section 8.3, we adapt the protocol of Section 4 to the evaluation of trust values.

8.1 Aggregation of Trust

For us, as in [28, 27, 20], 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 $uncertainty = 1 - trust - distrust$, it is sufficient to express trust with two values as $\langle trust, distrust \rangle$. In the following, we thus 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.

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.

Definition 18 (Sequential aggregation of trust)**.** *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* $\bigstar : \mathbb{D}^2 \times \mathbb{D}^2 \to \mathbb{D}^2$*, that computes the trust value over the trust path* $P_1 \overset{\langle a,b\rangle}{\to} P_2 \overset{\langle c,d\rangle}{\to} P_3$ *as*

$$\langle a,b\rangle \bigstar \langle c,d\rangle = \langle ac+bd, ad+bc\rangle.$$

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:

Definition 19 (Parallel aggregation of trust)**.** *For two disjoint paths* $P_1 \overset{\langle a,b\rangle}{\to} P_3$ *and* $P_1 \overset{\langle c,d\rangle}{\to} P_3$*, the associated* parallel aggregation of trust *is a function* $\maltese : \mathbb{D}^2 \times \mathbb{D}^2 \to \mathbb{D}^2$*, that computes the resulting trust value as:*

$$\langle a,b\rangle \maltese \langle c,d\rangle = \langle a+c-ac, bd\rangle.$$

Note that $\maltese$ is a commutative operator, since $\langle a,b\rangle \maltese \langle c,d\rangle = \langle a+c-ac, bd\rangle = \langle c+a-ca, db\rangle = \langle c,d\rangle \maltese \langle a,b\rangle$. We prove the following Lemma.

Lemma 20. *$\langle a,b\rangle$ is invertible for $\maltese$ if and only if (b is invertible in $\mathbb{D}$) and ($a = 0$ or $a-1$ is invertible).*

Proof. As $\langle a+0-a.0,\ b.1\rangle = \langle a,b\rangle$, we have that $\langle 0,1\rangle$ is neutral for $\maltese$. 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,

$$\begin{aligned}\langle a(a-1)^{-1}, b^{-1}\rangle \maltese \langle a,b\rangle &= \langle a,b\rangle \maltese \langle a(a-1)^{-1}, b^{-1}\rangle \\ &= \langle a + a(a-1)^{-1} - a^2(a-1)^{-1}, bb^{-1}\rangle \\ &= \langle 0,1\rangle.\end{aligned}$$

Similarly, if $\langle a,b\rangle \maltese \langle c,d\rangle = \langle 0,1\rangle$, 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. □

8.2 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:

$$E(\langle a,b\rangle) = \langle E(a), E(b)\rangle. \tag{6}$$

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

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

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 the following equalities: $E(a)^c E(b)^d = E(ac+bd)$, $E(a)^d E(b)^c = E(ad+bc)$, as well as $E(a)E(c)E(a)^{-c} = E(a+c+a(-c))$ 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 Note that $E(c)$ can be computed with the public key, from c. □

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

8.3 Multi-party Private Trust Evaluation

We now show how to fully adapt the protocol of Section 4 to the evaluation of trust values with parallel and sequential aggregations:

Corollary 22. *The protocol DSDP of Algorithm 2 can be applied on trust values, provided that the random values r_i are invertible for $\maltese$.*

Proof.

- u_i, v_i, r_i, c_i, α_i, β_i, Δ_i, γ 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 $E(\langle a,b\rangle) = \langle E(a), E(b)\rangle$;
- α_i is $E((u_i \bigstar v_i) \maltese r_i) = \mathrm{Add}(\mathrm{Mul}(E(v_i); u_i); r_i)$, 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, $\beta_i = E(\alpha_i \maltese \Delta_i) = \mathrm{Add}(E(\alpha_i); \Delta_i)$.
- Finally, as $\maltese$ is commutative, S is recovered by adding the inverses for $\maltese$ of the r_i.

□

From [20, 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 9, it suffices to set s to the neutral element of $\maltese$ (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 $\lfloor x 2^p \rfloor \mod N$ from $[0,1] \to \mathbb{D}$. Then the dot-product can be bounded as follows:

Lemma 23. *If each coefficient of the u_i and v_i are between 0 and $2^p - 1$, then the coefficients of $S = \maltese_{i=1}^{n} (u_i \bigstar v_i)$ are bounded by $2^{n(2p+1)}$ in absolute value.*

Proof. For all u, v, the coefficients of $(u \bigstar v)$ 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 $\maltese$, the absolute values of the coefficients remain less than $2^{k(2p+1)} - 1$. □

Therefore, with N an 2048 bits modulus and $n \leq 4$ in the *ESDP* protocols of Algorithm 9, Lemma 23 allows a precision close to $2^{-255} \approx 10^{-77}$.

In conclusion, we provide an efficient and secure protocol $DSDP_n$ 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. As shown

in [27] the calculus of trust can directly be applied to public-key infrastructures.

Thus, for instance, our matrix-multiplication protocols can be used to compute the trust aggregation between certification authorities (CA) in public-key infrastructures (PKI). In [22], they identify more than 1800 certificates for these entities, controlled by around 700 different organizations. Thus, we ran a parallel execution of *YTP-SS* in the manner of $PDSMM_n$ (see Section 6) with 700 players. With a simple trust metric, represented by only trust and uncertainty, we needed less than 430 seconds to compute the full 700×700 private multiparty matrix multiplication. In the end, each certification authorities gets a global trust view of their neighbors in the network, using the other authorities' recommendations in about 7 minutes. A few more iterations might be needed to get a more accurate view, but in any case this shows that frequently updating a trust evaluation via a distributed and private matrix multiplication is nowadays practical for a large number of certification authorities.

9 Conclusion and Perspective

In this article, we provide two dual protocols *DSDP* and *YTP-SS*, to securely compute dot products (against semi-honest adversaries) in the MPC model, with unusual data division between n players. Both protocols complexity in communications and computations are bounded by $O(n)$. In parallel, they require less than five communication steps. They also have a computational depth bounded by $O(\log n)$ and can be used to perform a private matrix multiplication. The complexity bounds of the different protocols developed in this paper are recalled in Table 1. We give the overall volume of communication, from cubic in [17] to quadratic (§ 3) and then linear for *DSDP* and *YTP-SS*. We also give the minimal number of rounds required to exchange this amount of data, as well as if Paillier-like encryption (as opposed to Benaloh-like encryption) can be used within the protocols. For *DSDP*, the number of rounds can be decreased from linear to constant, using the parallel method of Alg. 9, in the honest-but-curious setting (H). Similarly, *YTP-SS* has a linear number of rounds which can be reduced to constant, using the parallel method of Alg. 9. Then, in the malicious setting (M), the security of both schemes can be enhanced with the random-ring mitigation scheme, either on average or in the worst-case (Algorithm 11, full "wiretap" variant).

Table 1: Communication complexity bounds

Protocol	Volume	Rounds	Paillier	Security
MPWP	$O(n^3)$	$O(n)$	✗	?
P-MPWP (§ 3)	$n^{2+o(1)}$	$O(n)$	✓	?
Alg. 2 ($DSDP_n$)	$n^{1+o(1)}$	$O(n)$	✓	H
Alg. 9 ($PDSMM_n$)	$n^{1+o(1)}$	5	✓	H
Alg. 15 ($YTP\text{-}SS_n$)	$n^{1+o(1)}$	$O(n)$	✓	H
Alg. 15 (parallel)	$n^{1+o(1)}$	5	✓	H
Alg. 11 over Alg. 9 or Alg. 15 (Average)	$n^{1+o(1)}$	5	✓	M
Alg. 11 over Alg. 9 or Alg. 15 (Wiretap)	$n^{2+o(1)}\ln\left(\frac{1}{\varepsilon}\right)$	5	✓	M

These protocols can also be adapted to securely compute trust aggregation between players, defined on an unclassical matrix product, where base operations are replaced

with ones defined on specific monoids.

Then, to further improve running times we plan to see if it is possible to replace Paillier's encryption with Schmidt-Samoa and Takagi's scheme that can combine the advantages of Benaloh-like schemes and those of Paillier-like ones. Another line of research would be to be able to use fast matrix multiplication instead. There, existing practical algorithms work recursively, in blocks, and with pre- and post-additions. Therefore the players would have to enter some private addition schemes and deal with recursion.

Besides, we used automatic protocol verification tools to prove the security of the protocols against semi-honest adversaries and find attacks against malicious ones. Then, we developed countermeasures for each attacks discovered by the tool. In particular, we use proofs of knowledge checks and a random ring strategy which reduces drastically the probability of attacks when the protocol is repeated, even a small number of times.

Our random ring order mitigation scheme is actually quite generic and allows us to ensure the security against a fixed number of malicious participants. One possible extension would be to consider more powerful intruders, for instance adaptative intruders that can corrupt some participants only during some phases of the protocols and according to the chosen random ring order. In this case, we need to choose the random ring order in a more secure manner in order to prevent the intruder from anticipating it. Then, one possible solution would be to use a certified time stamp signed by a trusted time-stamping authority in the computation of the random ring. With this method the intruder cannot anticipate the random ring order and should not be able to mount an adaptative attack.

References

[1] Artak Amirbekyan and Vladimir Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *AusDM 2007*, volume 70 of *CRPIT*, pages 209–214, 2007. URL: http://crpit.com/confpapers/CRPITV70Amirbekyan.pdf.

[2] Myrto Arapinis, Stéphanie Delaune, and Steve Kremer. Dynamic tags for security protocols. *Logical Methods in Computer Science*, 10(2:11), June 2014. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/ADK-lmcs14.pdf, doi:10.2168/LMCS-10(2:11)2014.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, Michael R., J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. of CAV'2005*, LNCS 3576, pages 281–285. Springer, 2005. URL: https://www.inf.ethz.ch/personal/basin/pubs/avispa05.pdf.

[4] M. J. Atallah, H. G. Elmongui, V. Deshpande, and L. B. Schwarz. Secure supply-chain protocols. In *EEE International Conference on E-Commerce (CEC'03), 2003*, pages 293–302, June 2003. doi:10.1109/COEC.2003.1210264.

[5] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. In Frank Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms*

and Data Structures: 7th International Workshop, WADS 2001 Providence, RI, USA, August 8–10, 2001 Proceedings, pages 165–179, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-44634-6_16.

[6] David Basin, Sebastian Mödersheim, and Luca Viganò. An on-the-fly model-checker for security protocol analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security – ESORICS 2003: 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003. Proceedings*, pages 253–270, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/978-3-540-39650-5_15.

[7] Necdet Batir. Sharp bounds for the psi function and harmonic numbers. *Mathematical inequalities and applications*, 14(4), 2011. doi:10.7153/mia-14-77.

[8] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, STOC'88, pages 1–10. ACM, 1988. doi:10.1145/62212.62213.

[9] Josh Benaloh. Dense probabilistic encryption. In *First Annual Workshop on Selected Areas in Cryptography*, pages 120–128, Kingston, ON, May 1994. URL: http://sacworkshop.org/proc/SAC_94_006.pdf.

[10] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In KennethG. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-20465-4_11.

[11] B. Blanchet. *Cryptographic Protocol Verifier User Manual*, 2004. URL: http://www.di.ens.fr/~blanchet/crypto/proverif-manual.ps.gz.

[12] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96, Cape Breton, Canada, 2001. IEEE Comp. Soc. Press. doi:10.1109/CSFW.2001.930138.

[13] David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf, and René Peralta. Demonstrating possession of a discrete logarithm without revealing it. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 1986. doi:10.1007/3-540-47721-7_14.

[14] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, December 2002. doi:10.1145/772862.772867.

[15] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-32009-5_38.

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

[17] Shlomi Dolev, Niv Gilboa, and Marina Kopeetsky. Computing multi-party trust privately: in $O(n)$ time units sending one (possibly large) message at a time. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1460–1465, New York, NY, USA, 2010. ACM. doi:10.1145/1774088.1774401.

[18] Wenliang Du and M. J. Atallah. Privacy-preserving cooperative statistical analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference*, ACSAC '01, pages 102–110, December 2001. doi:10.1109/ACSAC.2001.991526.

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

[20] Jean-Guillaume Dumas and Hicham Hossayni. Matrix powers algorithm for trust evaluation in PKI architectures. In Audun Jøsang, Pierangela Samarati, and Marinella Petrocchi, editors, *STM'2012, Proceedings of the eigth International Workshop on Security and Trust Management (co-ESORICS 2012), Pisa, Italy*, volume 7783 of *Lecture Notes in Computer Science*, pages 129–144, September 2012. doi:10.1007/978-3-642-38004-4_9.

[21] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Private multi-party matrix multiplication and trust computations. In Pierangela Samarati, editor, *13th International Conference on Security and Cryptography (SECRYPT 2016)*, pages 61–72, July 2016. doi:10.5220/0005957200610072.

[22] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 291–304, New York, NY, USA, 2013. ACM. doi:10.1145/2504730.2504755.

[23] Simon N. Foley, Wayne Mac Adams, and Barry O'Sullivan. Aggregating trust using triangular norms in the keynote trust management system. In Jorge Cuéllar, Javier Lopez, Gilles Barthe, and Alexander Pretschner, editors, *Security and Trust Management - 6th International Workshop, STM 2010, Athens, Greece, September 23-24, 2010, Revised Selected Papers*, volume 6710 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2010. doi:10.1007/978-3-642-22444-7_7.

[24] Laurent Fousse, Pascal Lafourcade, and Mohamed Alnuaimi. Benaloh's dense probabilistic encryption revisited. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 2011. doi:10.1007/978-3-642-21969-6_22.

[25] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In Choon-sik Park and Seongtaek Chee, editors, *Information Security and Cryptology ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 104–120. Springer Berlin Heidelberg, 2005. doi:10.1007/11496618_9.

[26] Ramanathan V. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 403–412. ACM, 2004. doi:10.1145/988672.988727.

[27] Jingwei Huang and David M. Nicol. A formal-semantics-based calculus of trust. *IEEE Internet Computing*, 14(5):38–46, 2010. doi:10.1109/MIC.2010.83.

[28] Audun Jøsang. Probabilistic logic under uncertainty. In Joachim Gudmundsson and C. Barry Jay, editors, *Theory of Computing 2007. Proceedings of the Thirteenth Computing: The Australasian Theory Symposium (CATS2007). January 30 - Febuary 2, 2007, Ballarat, Victoria, Australia, Proceedings*, volume 65 of *CRPIT*, pages 101–110. Australian Computer Society, 2007. URL: http://crpit.com/abstracts/CRPITV65Josang.html.

[29] Pascal Lafourcade and Maxime Puys. Performance evaluations of cryptographic protocols. verification tools dealing with algebraic properties. In *FPS 2015*, 2015. doi:10.1007/978-3-319-30303-1_9.

[30] Yehuda Lindell. Secure computation for privacy preserving data mining. In John Wang, editor, *Encyclopedia of Data Warehousing and Mining, Second Edition (4 Volumes)*, pages 1747–1752. IGI Global, 2009. doi:10.4018/978-1-60566-010-3.

[31] Antonis Michalas, Tassos Dimitriou, Thanassis Giannetsos, Nikos Komninos, and Neeli R. Prasad. Vulnerabilities of decentralized additive reputation systems regarding the privacy of individual votes. *Wireless Personal Commnications*, 66(3):559–575, 2012. doi:10.1007/s11277-012-0734-z.

[32] Payman Mohassel. Efficient and secure delegation of linear algebra. *IACR Cryptology ePrint Archive*, 2011:605, 2011. URL: http://eprint.iacr.org/2011/605.

[33] Lawrence H. Ozarow and Aaron D. Wyner. Wire-tap channel II. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *EUROCRYPT'84, Paris, France*, volume 209 of *LNCS*, pages 33–50. Springer, 1984. doi:10.1007/3-540-39757-4_5.

[34] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999. doi:10.1007/3-540-48910-X_16.

[35] Bruce Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[36] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. doi:10.1145/359168.359176.

[37] Samuel S. Shepard, Renren Dong, Ray Kresman, and Larry Dunning. Anonymous id assignment and opt-out. In Sio-Iong Ao and Len Gelman, editors, *Electronic Engineering and Computing Technology*, pages 419–431, Dordrecht, 2010. Springer Netherlands. doi:10.1007/978-90-481-8776-8_36.

[38] Mathieu Turuani. The CL-Atse Protocol Analyser. In Frank Pfenning, editor, *17th International Conference on Term Rewriting and Applications - RTA 2006 Lecture Notes in Computer Science*, volume 4098 of *LNCS*, pages 277–286, Seattle, USA, August 2006. Springer. URL: https://hal.inria.fr/inria-00103573/document.

[39] I-Cheng Wang, Chih hao Shen, Tsan sheng Hsu, Churn-Chung Liao, Da-Wei Wang, and J. Zhan. Towards empirical aspects of secure scalar product. In *Information Security and Assurance, 2008. ISA 2008. International Conference on*, pages 573–578, April 2008. doi:10.1109/ISA.2008.78.

[40] Andrew C. Yao. Protocols for secure computations. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:160–164, 1982. doi:10.1109/SFCS.1982.88.

[41] Danfeng Yao, Roberto Tamassia, and Seth Proctor. Private distributed scalar product protocol with application to privacy-preserving computation of trust. In Sandro Etalle and Stephen Marsh, editors, *Trust Management: Proceedings of IFIPTM 2007: Joint iTrust and PST Conferences on Privacy, Trust Management and Security, July 30– August 2, 2007, New Brunswick, Canada*, pages 1–16, Boston, MA, 2007. Springer US. doi:10.1007/978-0-387-73655-6_1.