Outsourcing Scalar Products and Matrix Products on Privacy-Protected Unencrypted Data Stored in Untrusted Clouds

Josep Domingo-Ferrer[a], Sara Ricci[a], Carles Domingo-Enrich

[a] *Universitat Rovira i Virgili, Department of Computer Engineering and Mathematics, UNESCO Chair in Data Privacy, Av Països Catalans 26, E-43007 Tarragona, Catalonia*
{josep.domingo,sara.ricci}@urv.cat, cdomingo96@gmail.com

Abstract

Data controllers accumulate more and more data on people, of which a substantial proportion are personally identifiable and hence sensitive data. Storing and processing those data in local premises is increasingly inconvenient, but resorting to cloud storage and processing raises security and privacy issues. We show how to use untrusted clouds to compute scalar products and matrix products on privacy-protected data stored in them. These operations are useful in statistics, linear algebra, data analysis and engineering. In our solutions, the privacy-protected sensitive data stored in the clouds are not encrypted, but preserve some utility (that is, some statistical properties) of the original data. We consider two variants of honest-but-curious clouds: clouds that do not share information with each other and clouds that may collude by sharing information with each other. In addition to analyzing the security of the proposed protocols, we also evaluate their performance against a baseline consisting of downloading plus local computation.

Keywords: Privacy, cloud computing, data splitting, scalar product, matrix product, honest-but-curious clouds.

1. Introduction

With the advancement and spread of computation and communication technologies, the amount of data collected and stored by private and public sectors is constantly increasing. Storing and processing such huge amounts of information in local premises becomes very problematic, due to soaring costs of software, hardware, energy and maintenance. In this context, the need emerges to find a fast and cost-effective alternative. An attractive possibility for a data controller is to outsource storage and processing to a cloud [2]. Such outsourcing brings several benefits like elimination of infrastructure costs (no software/hardware investments needed), flexibility (storage and computing power can scale depending on business growth) and energy savings.

Unfortunately, storing and processing data in the cloud has also downsides related to security and privacy. A lot of the information being collected is personally identifiable and therefore sensitive. Neither the data controller nor the subjects to whom the data refer want the cloud service provider (CSP) to read, use or sell their data.

In this article we discuss several procedures to store *and process* sensitive data in a privacy-preserving way in untrusted clouds, where processing consists of two basic operations: scalar products and matrix products. These operations are useful in statistics and data analysis (to compute correlations between attributes, contingency tables, etc.), and also in engineering (image encryption, 3D graphics simulation, etc.); see Section 1.2 of [23] and references therein. In our solutions, *the privacy-protected sensitive data stored in the clouds are not encrypted* and *preserve some of the utility (that is, some statistical features) of the original data*. This allows making the most of the outsourced data, while ensuring that no original records can be re-created from the outsourced records. The outsourced data can be used for purposes other than computing scalar products or matrix products. This is a relevant difference with respect to related work on algebraic computation outsourcing (see Section 3.2).

Specifically, the outsourced data preserve means and standard deviations of attributes for the entire data set and even in subdomains of it. In some of our protocols, we use vertical splitting, so that each cloud stores a cleartext fragment on which any statistical analysis can be directly performed by the cloud. In the rest of our protocols, the cloud stores synthetic or anonymized versions of the original data set that preserve the above mentioned statistics. The goal is that the outsourced version retains some utility for exploratory analysis by any user with direct access to the cloud-stored data (who should however be unable to reconstruct the original data). Note that there are many organizations interested in releasing privacy-protected/anonymized data for secondary analysis, including but not limited to official statistics [19].

Following the architecture defined in the "CLARUS" European H2020 project [9] (within which this work has been carried out), we will assume a proxy located in a domain trusted by the data controller (e.g., a server in her company's intranet or a plug-in in her device) that implements security and privacy-enabling features towards the cloud service providers. We will call this trusted proxy CLARUS.

Contribution and plan of this paper

In [5], we evaluated some non-cryptographic proposals for secure scalar product and secure matrix product on vertically split data presented in the literature. We then proposed and enhanced some protocols adapted to the CLARUS scenario. The CSPs were assumed to be honest-but-curious and not sharing information with each other.

Here, we start from the conclusions of that previous work and break new ground by: i) exploring new non-cryptographic protocols for the scalar product on split data; ii) considering also cryptographic protocols to compute on split

data; and iii) relaxing the non-sharing assumption. In the cryptographic protocols we use encryption only in the communication between clouds; however, the sensitive data stored in the clouds are protected by splitting, not by encryption. Regarding the sharing assumptions, we first assume that the CSPs do not pool their fragments to reconstruct the original data set; we start from two existing protocols for this setting, one without cryptography and one using cryptography, and we present two new non-cryptographic protocols and two variants of the cryptographic protocol. We then relax the non-sharing assumption and present two additional non-cryptographic protocols that are sharing-resistant, even though they require substantial cloud storage (because they rely on data replication rather than splitting).

This paper is organized as follows:

- Section 2 presents the CLARUS architecture and the security models considered in the rest of the paper for the untrusted CSPs: i) honest-but-curious CSPs not sharing information with each other; ii) honest-but-curious CSPs that may collude by sharing information with each other, but that lack side knowledge on the original data set; iii) honest-but-curious CSPs sharing information with each other and having side knowledge on the original data set.

- Section 3 reviews background on data splitting and related work on algebraic computation outsourcing and secure scalar products.

- Section 4 focuses on secure scalar products on vertically partitioned data when CSPs are honest-but-curious and do not share information: a non-cryptographic protocol and a cryptographic protocol are reviewed, and then two new non-cryptographic protocols and two variants of the cryptographic protocol are presented. After that, we show how secure scalar products between pairs of clouds can be combined with computations involving a single cloud to perform data analyses such as correlations and contingency tables.

- Section 5 presents a new protocol that can resist information sharing between CSPs but assumes the CSPs have no side knowledge about the original data set; rather than computing individual scalar products, this protocol computes a matrix product $\mathbf{X}^T\mathbf{X}$ of an original data set $\mathbf{X}$ of which the cloud only knows a masked version $\mathbf{Y}$. Just like the scalar products allowed computing the data set correlation matrix, so does the above matrix product; furthermore, the clouds can also be used to compute the means and the standard deviations of attributes in $\mathbf{X}$.

- Section 6 proposes another new sharing-resistant protocol to compute the matrix product $\mathbf{X}^T\mathbf{X}$ that involves heavier computation but which stays safe even if the CSPs have information on the statistical structure of the original data set $\mathbf{X}$.

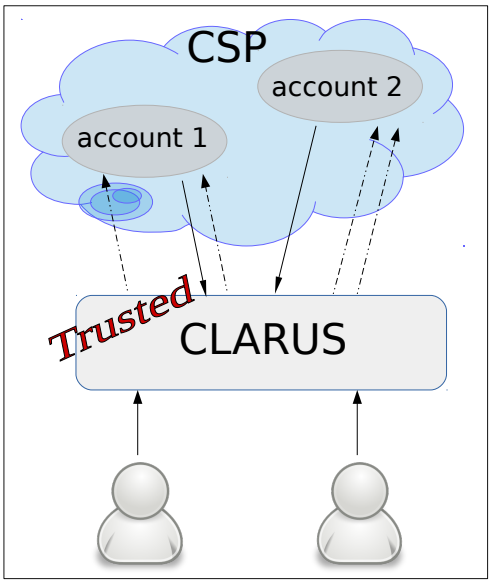

Figure 1: CLARUS is a proxy trusted by the data controller and the controller's users that implements security and privacy-enabling features towards the untrusted cloud service provider

- In Section 7, the computation and communication costs of all protocols described in Sections 4, 5 and 6 are assessed and compared against a benchmark protocol consisting of the CLARUS proxy downloading the entire data set and locally computing on the downloaded data set.
- Section 8 presents the experimental results obtained by implementing the proposed protocols in a multi-cloud scenario.
- Finally, Section 9 lists some conclusions and future research lines.

The Appendix contains mathematical technicalities related to the cryptographic protocol in Section 4 and the sharing-resistant protocol in Section 5.

2. CLARUS architecture, assumptions and security models

As shown in Figure 1, CLARUS is a proxy located in a domain trusted by the data controller that implements security and privacy-enabling features towards the cloud service provider so that i) the CSP only receives privacy-protected versions of the controller's (or the controller's users') data, ii) CLARUS makes the access to such data transparent to the controller's users (by adapting their queries and reconstructing the results retrieved from the cloud) and iii) it remains possible for the users to leverage the cloud to perform accurate computations on the outsourced data without downloading them.

The *raison d'être* of such an architecture is to outsource as much storage and computation as possible to the cloud in a privacy-preserving manner, and keep the computational load of the CLARUS proxy (sitting in the controller's premises) as low as possible. There is an underlying assumption that computing in the cloud is cheaper and/or otherwise more convenient than computing in the controller's local facilities. To evaluate the performance of a protocol, we will focus on how much it reduces the work to be done by the CLARUS proxy compared to the trivial alternative of CLARUS downloading the entire data set, unprotecting it and computing locally on the downloaded unprotected data.

Regarding privacy and security, the CSP is not trusted and hence it is not given access to the entire original data set. The CSP just sees fragments of the original data set or an anonymized version of it. More specifically, we consider three different security models for the CSPs, depending on their information sharing and their level of background knowledge:

Honest-but-curious, non-sharing CSPs. The CSPs honestly fill their role in the protocols, and they do not share information with each other (perhaps because they do not even know each other). In particular, they do not pool together the data fragments they hold. However, each CSP may be curious to infer and analyze the data it stores and the message flows received during the protocol, in order to acquire additional information. This model is common in the cloud computing literature, e.g. see [6].

Honest-but-curious, sharing CSPs without background knowledge. We relax here the previous assumption about non-sharing: while honestly following the protocol, CSPs may collude by sharing information with each other. However, we assume they do not have any initial side knowledge about the original data set.

Honest-but-curious, sharing CSPs with side knowledge. This is the most demanding model we consider. In addition to sharing information with each other, the CSPs have background knowledge on the statistical structure of the original data set. Nevertheless, they honestly perform their roles in the protocols.

We do not consider malicious CSPs that may deviate from the protocols because they would not be very useful for computation outsourcing in our context. The data controller/owner using the CLARUS proxy is assumed to rent the CSPs to get help from them. Hence, it would be pointless if it took CLARUS more effort to check that malicious CSPs carry out the computations correctly than to download the data and do the computations locally.

3. Background and related work

In order to ensure privacy, one might think of encrypting sensitive data before storing them in the cloud. However, encrypting data substantially hampers processing them in the cloud (i.e., using the cloud's computing power to perform

queries and compute some mathematical operations on the outsourced data). Admittedly, searchable (e.g. [21]) and homomorphic (e.g. [27]) encryption also allow carrying out some operations on ciphertext [14], but with substantial functionality and performance restrictions (only some operations are feasible and special methods must be used for them). Furthermore, computing on encrypted data is extremely costly [28], and it requires careful management of encryption keys: the proxy ought to locally store all keys used.

Another approach to protecting sensitive data is to split the data set into several fragments and store them in different clouds (each provided by a different CSP) or in different cloud accounts with the same CSP. We review background on data splitting in what follows.

3.1. Data splitting

Data splitting (or data partitioning or fragmentation) means dividing an original sensitive data set into fragments and storing each fragment in a different site, in such a way that the fragment in any site considered in isolation is no longer sensitive. Data splitting has long been used as a privacy-preserving technique [30, 31, 10].

Queries on split data can often be answered much more efficiently than queries on encrypted data (see [1]). In data splitting, the most challenging step is usually to efficiently compute on the fragmented data when the computations involve more than one fragment: in this case, the clouds may need to exchange (part of) their respective fragments, but none of them ought to reveal its own private information. Specifically, challenging tasks in computing on split/distributed data are data mining [33] and data correlation [34]. The literature on parallel processing for statistical computation has partly treated this topic: the way to combine partial results obtained from independent processors may provide guidance on how to treat distributed data. On the other hand, privacy-preserving data mining on partitioned data can be of use too, as its main objective is to mine data owned by different parties who are willing to collaborate in order to get better results, but who do not want or cannot share their raw original data.

If properly performed, fragmentation does not allow linking confidential information to specific individuals. For example, if a fragment consists of the values of an attribute "Diagnosis", then clearly just knowing a list of diagnoses is useless to an intruder, because he cannot associate them to the corresponding subjects (it is nearly as useless as seeing a list of diseases and their frequencies in a manual of medicine).

Splitting can be horizontal (each fragment contains the values of all attributes but only for a subset of subjects), vertical (each fragment contains the values of a subset of attributes for all subjects), or mixed (each fragment contains the values of a subset of attributes for a subset of subjects). Whereas vertical and mixed splitting preserve privacy by decomposition, horizontal splitting does not do so because all the information on the same individual subject is stored together [1]. Hence, if preserving anonymity and preventing attribute disclosure is a key concern, horizontal partitioning is not suitable.

Vertical splitting methods were proposed in [1] and [16] to ensure confidentiality. Both methods rely on predefined constraints describing risky attribute combinations. An example of risky pair is passport number and disease, whereas blood pressure and disease is generally a safe pair. The goal of both methods is to partition the original data set into two vertical fragments in such a way that, if some risky attributes are stored together in a fragment, some of them need to be encoded/encrypted. In [7] the authors illustrate a similar approach to [1] and [16] but using an arbitrary number of non-linkable data fragments, which can be stored at an arbitrary number of providers. Also, [8] presents a solution for splitting data in two vertical fragments without requiring the use of encryption, but rather using a trusted party (the owner) to store a portion of the data and perform part of the computation.

Regarding mixed splitting, in general, it does not improve on vertical splitting in terms of privacy and it complicates distributed computation (no local computation on entire attributes is possible any more at the locations holding fragments).

In summary, vertical splitting is the most suitable splitting for privacy, since in statistical databases it is the *joint* distribution of several attributes that is sensitive (because it may lead to re-identification of the subject behind a tuple of values). Attributes in isolation (or even groups of attributes whose value combinations are very common) are not sensitive, as illustrated by the "Diagnosis" example above.

When storing dynamically changing sensitive data in the cloud, vertical splitting is very convenient: each time one wants to add/update a record, this can be done separately in each fragment, which preserves the isolation inherent to splitting. Also, additions/updates are fast, because nothing needs to be done to the rest of records (those that do not change). On the contrary, if data stored in the cloud are encrypted or masked rather than split, any record addition/update requires re-encrypting or re-anonymizing the original data set including the added/updated record and re-uploading the entire re-encrypted or re-anonymized data set. This is clearly more costly. Furthermore, if the fragments are stored at different CSPs and these do not share information, splitting is more privacy-preserving than masking for dynamic data, because in masking the (single) CSP might infer the value of some original records by comparing the successive anonymized versions of the data set.

3.2. Related work on algebraic computation outsourcing

There is a substantial amount of literature devoted to outsourcing matrix and polynomial computations. We next review it and then highlight the differences with our approach.

In [3], a client securely outsources algebraic computations to one or several remote servers, in such a way that the server learns nothing about the client's private input or the result of the computation, and any attempted corruption of the answer by the server is detected with high probability. This scheme is based on multiparty secure computation via secret sharing. In [24], a client outsources a matrix inversion to an untrusted cloud, so that the cloud does not

learn either the original or the inverted matrices. Furthermore, the protocol is resistant against a cheating cloud. In [23], the authors present a protocol to outsource multiplication of large matrices that can detect cheating by the server. The more recent contribution [29] follows the same line (outsourcing polynomials and matrix computations), but it focuses on public verifiability of the computation (any third party can verify its correctness, not only the client as in the previous proposals). This comes at the price of using more complex cryptographic schemes. In all the schemes reviewed in this paragraph, the server(s) only see(s) encrypted versions or shares of the actual data: the client must create these encrypted versions for each protocol execution and decrypt the final result.

Outsourcing matrix computations where the server computes on additively split matrices rather than encrypted matrices is considered in [26]. Even tough no encryption is used, the split versions of the matrics seen by the server do not preserve any of the statistical features of the original data (they look gibberish), so that no direct exploratory analyses can be performed on them.

A substantial difference between our proposals in this paper and the previous literature is that we assume that the cloud(s) compute on data that have been previously outsourced and privacy-protected *not only* to compute scalar products or matrix products on them. Specifically, the outsourced data preserve some of the utility of the original data, as explained in Section 1 above.

A second difference is that, as mentioned at the end of Section 2, in the CLARUS setting we can assume that CSPs are not malicious: hence, we do not need all the cryptographic apparatus of the above cryptographic proposals to detect cheating.

3.3. Related work on secure scalar products

Secure scalar products can be based on cryptography or not. Cryptographic approaches may use a variety of techniques. For instance, the protocol in [18] involves homomorphic encryption. Similarly, [25] present a protocol in which all users encrypt their private vector using fully homomorphic encryption and upload it to a server. A user (initiator) sends a scalar product query to the server, which returns the final result after a series of computations and collaborative operations with the other user.

Non-cryptographic approaches are rather based on modifying the data before sharing them in such a way that the original data cannot be deduced from the shared data but the final results are preserved (e.g. [10], [12], [13], [22]).

The vast majority of protocols proposed for computing on vertically split data do without a trusted third party (with the exception of the commodity server solution of [12, 13]). While avoiding a trusted third party is technically elegant, it normally takes more computation and communication. See [32] for a performance comparison of several protocols for the secure scalar product.

As explained in Section 2, our scenario involves a trusted party, the CLARUS proxy; in particular, the clouds holding the private vectors do not see the result of the scalar product (that is only seen by the CLARUS proxy). Hence,

our setting is simpler than the one usually assumed in secure two-party computation, in which there is no third party and at least one of the parties learns the computation output.

4. Secure scalar product for data analysis on vertically partitioned data

Initially, the data are stored by the data controller vertically split across several clouds, either directly or via the CLARUS proxy. After that, users want to use the CLARUS proxy to compute scalar products between attributes stored in different clouds. Our goal is to minimize the computation at the CLARUS proxy, which is a resource in the controller's premises. In contrast, we assume that the CSPs have unlimited storage and computational power, and, therefore, we want to shift as much of the computational and storage load as possible to the CSPs' side. As to security, *no cloud should learn the data stored by the other clouds, but the CLARUS proxy is trusted and is entitled to know everything.*

In Section 4.1, we start from the most efficient protocol for computing on vertically split data identified in [32] and adapted for use with the CLARUS proxy in [5]. We identify several shortcomings of this protocol in the CLARUS setting and we modify it by replacing or complementing random noise additions with permutations. In Section 4.2, we revise and adapt to the CLARUS scenario a cryptographic protocol for the secure distributed scalar product. Section 4.3 illustrates how secure scalar products can be used to compute correlations and contingency tables on vertically split data.

4.1. Secure scalar product without cryptography

In this section, we work under the honest-but-curious and non-sharing model: the CSPs neither deviate from the protocols nor pool the data fragments they hold. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors with n components owned by Alice and Bob (who can be two CSPs), respectively. The goal is to securely compute the product $\boldsymbol{x}^T\boldsymbol{y}$, see Figure 2. The privacy of the following protocols relies on the fact that the original vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are not shared at any time by the respective CSPs owning them; only linear transformations of them are, such that the number of unknowns (randomness) added by the tranformations is greater than or equal to the number of private unknowns. In the following protocols, CLARUS obtains the desired result; note than any disclosure by Alice or Bob to CLARUS does not entail any privacy leak, because (unlike Alice and Bob) CLARUS is trusted by the data controller/owner.

We take as starting point the protocol proposed in [12, 13], that is based on what they call a commodity server. This protocol is identified as the most efficient one in [32]. Let Alice and Bob be as previously defined and let a third, non-sharing cloud Charlie play the role of the commodity server. In [5], we suggested the following variant adapted for use with the CLARUS proxy:

Protocol 1.

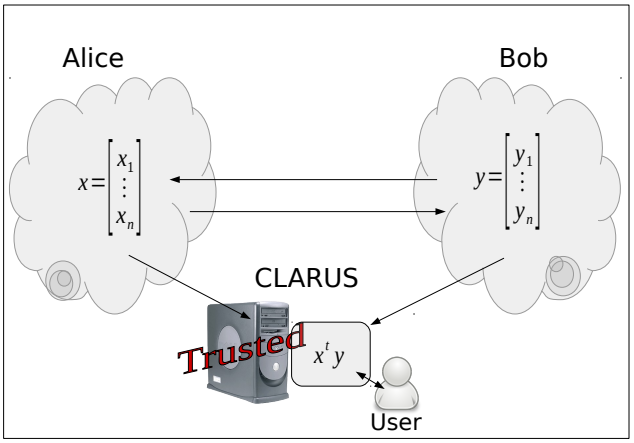

Figure 2: Honest-but-curious, non-sharing CSPs

1. *Charlie generates a random n-vector $\boldsymbol{r}_x$ and another n-vector $\boldsymbol{r}_y$ and computes $p = \boldsymbol{r}_x^T \boldsymbol{r}_y$ (note that p is a number).*
2. *Charlie sends Alice the seed for a common random generator of $\boldsymbol{r}_x$, and sends Bob the seed for a common random generator of $\boldsymbol{r}_y$ (or equivalently Charlie sends $\boldsymbol{r}_x$ and $\boldsymbol{r}_y$ to Alice and Bob, respectively, if sending these vectors is faster than Alice and Bob generating them). Also, Charlie sends p to CLARUS.*
3. *Alice sends $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{r}_x$ to Bob.*
4. *Bob sends $t = \hat{\boldsymbol{x}}^T \boldsymbol{y}$ to CLARUS and sends $\hat{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{r}_y$ to Alice.*
5. *Alice sends $s_x = \boldsymbol{r}_x^T \hat{\boldsymbol{y}}$ to CLARUS.*
6. *CLARUS computes $t - s_x + p$ $[= (\boldsymbol{x} + \boldsymbol{r}_x)^T \boldsymbol{y} - \boldsymbol{r}_x^T (\boldsymbol{y} + \boldsymbol{r}_y) + \boldsymbol{r}_x^T \boldsymbol{r}_y] = \boldsymbol{x}^T \boldsymbol{y}$.*

The authors of [12] show that their basic protocol allows neither Alice to learn $\boldsymbol{y}$ nor Bob to learn $\boldsymbol{x}$. In [5], is it is shown that the above variant still offers the same security regarding Alice and Bob as the basic protocol [12].

It is important to note that in Protocol 1 (as well as in the basic protocol [12]), the generated random vectors $\boldsymbol{r}_x$ and/or $\boldsymbol{r}_y$ should be reused in successive instances of the protocol with the same original data vectors $\boldsymbol{x}$ and/or $\boldsymbol{y}$, in order to avoid leaking new equations that would facilitate the reconstruction of a player's original data vector by the other player. It is easy for Alice to store $\boldsymbol{r}_x$ along with $\boldsymbol{x}$ for subsequent potential reuse, and the same holds for Bob with respect to $\boldsymbol{r}_y$ and $\boldsymbol{y}$. However, computing p requires knowledge of *both* $\boldsymbol{r}_x$ and $\boldsymbol{r}_y$, which neither Alice nor Bob have. On the other hand, Charlie knew both random vectors when he generated them, but he is unaware of any reuse unless told. So we propose to add the following two preliminary steps to Protocol 1:

pi. If Alice wants to reuse a previous private vector $\boldsymbol{x}$, she sends Charlie the corresponding seed of the random vector $\boldsymbol{r}_x$ (or equivalently sends the vector if doing so is faster than Charlie generating it).

pii. If Bob wants to reuse a previous private vector $\boldsymbol{y}$, he sends Charlie the seed of $\boldsymbol{r}_y$ (or equivalently sends the vector if doing so is faster than Charlie generating it).

and modify the first two steps of Protocol 1 as

1. If Charlie has not received $\boldsymbol{r}_x$, he generates it randomly; if Charlie has not received $\boldsymbol{r}_y$, he generates it randomly. Charlie computes $p = \boldsymbol{r}_x^T \boldsymbol{r}_y$.

2. If Charlie generated $\boldsymbol{r}_x$, he sends the seed used to generate this vector to Alice; if Charlie generated $\boldsymbol{r}_y$, he sends the seed used to generate this vector to Bob (equivalently, instead of sending seeds, Charlie may send the actual vectors if doing so is faster). Also, Charlie sends p to CLARUS.

Beyond the need to manage reuse as specified above, using random rectors can result in the following potential weaknesses:

- Weak choices of $\boldsymbol{r}_x$ and $\boldsymbol{r}_y$ could also leak information, and should therefore be avoided: for example, an unsafe choice is when only one component of $\boldsymbol{r}_x$ (or $\boldsymbol{r}_y$) is different from zero.

- While the basic protocol in [12] and Protocol 1 do not leak the *exact* values of $\mathbf{y}$ to Alice or the exact values of $\mathbf{x}$ to Bob, it may be possible to infer the range of some elements in $\boldsymbol{x}$ and $\boldsymbol{y}$ from the range of the random vectors, $\boldsymbol{r}_x$ and $\boldsymbol{r}_y$. For example, imagine the elements of $\boldsymbol{x}$ are known to lie in the domain $[0, 100]$ (the domain of an attribute may sometimes be estimated from its semantics, e.g., the domain of *Age* can be estimated at $[0, 100]$). On the other hand, assume Charlie generates the elements of $\boldsymbol{r}_x$ by randomly sampling the $[0, 200]$ domain (the parameters of (pseudo)random number generation are normally public). Then, if an element of vector $\boldsymbol{x}+\boldsymbol{r}_x$ is 250, Bob learns that the corresponding original element in vector $\boldsymbol{x}$ is greater than 50.

A way to avoid the previous shortcomings of noise addition is to resort to the other main principle of non-cryptographic data protection, namely permutation [11]. Independent random permutation of the values of each attribute affords suitable protection if: a) the values taken by the attribute to be permuted are diverse enough; b) breaking the joint occurrence of attribute values in original records is deemed sufficient to protect the subjects' privacy, but the values of each attribute can be released as long as they cannot be linked to the corresponding subjects (in fact, using vertical data splitting for privacy is also predicated on this assumption). Specifically, we propose an alternative new permutation-based protocol, which is graphically depicted in Figure 3 and whose steps are as follows:

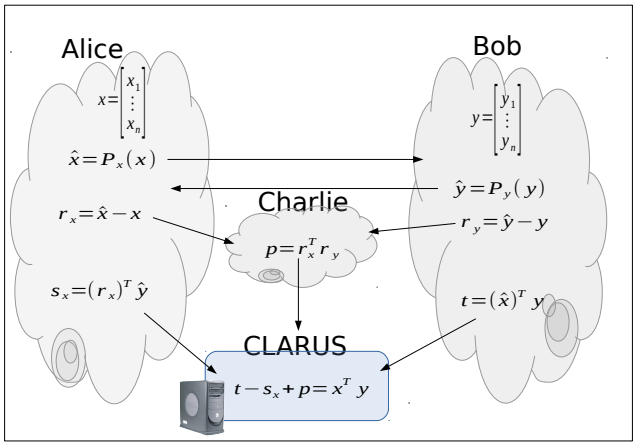

Figure 3: In Protocol 1.1, the CSPs permute their private vectors

Protocol 1.1.

1. *Alice randomly permutes the values in her private vector to obtain* $\hat{\boldsymbol{x}} = \mathcal{P}_x(\boldsymbol{x})$.
2. *Alice sends* $\hat{\boldsymbol{x}}$ *to Bob and* $\boldsymbol{r}_x = \hat{\boldsymbol{x}} - \boldsymbol{x}$ *to Charlie.*
3. *Bob randomly permutes the values in his private vector to obtain* $\hat{\boldsymbol{y}} = \mathcal{P}_y(\boldsymbol{y})$.
4. *Bob sends* $\hat{\boldsymbol{y}}$ *to Alice and* $\boldsymbol{r}_y = \hat{\boldsymbol{y}} - \boldsymbol{y}$ *to Charlie.*
5. *Charlie sends* $p = \boldsymbol{r}_x^T \boldsymbol{r}_y$ *(note that* p *is a number) to CLARUS.*
6. *Bob sends* $t = \hat{\boldsymbol{x}}^T \boldsymbol{y}$ *to CLARUS.*
7. *Alice sends* $s_x = \boldsymbol{r}_x^T \hat{\boldsymbol{y}}$ *to CLARUS.*
8. *CLARUS computes* $t - s_x + p \ \ [= (\boldsymbol{x} + \boldsymbol{r}_x)^T \boldsymbol{y} - \boldsymbol{r}_x^T (\boldsymbol{y} + \boldsymbol{r}_y) + \boldsymbol{r}_x^T \boldsymbol{r}_y] = \boldsymbol{x}^T \boldsymbol{y}$.

Unlike in Protocol 1, randomness in Protocol 1.1 does not consist of adding random numbers and it is not generated by Charlie: Alice and Bob are the ones generating random permutations for their own vectors. Hence, the shortcomings identified above for Protocol 1 (need to handle random vector reuse, possible weak choices of random values, possible partial inferences) do not apply to Protocol 1.1. Regarding security, we have the following result.

Proposition 1 (Security). *After participating in Protocol 1.1, Charlie does not learn* $\boldsymbol{x}$ *or* $\boldsymbol{y}$*; the probability of Bob's guessing* $\boldsymbol{x}$ *is at most*

$$\frac{n_1^x! n_2^x! \ldots n_{d_x}^x!}{n!},$$

where d_x is the number of different values among the n values of $\boldsymbol{x}$, and n_i^x is the number of repetitions of the i-different value; analogously, the probability of Alice's guessing $\boldsymbol{y}$ is at most

$$\frac{n_1^y! n_2^y! \ldots n_{d_y}^y!}{n!}.$$

Further, the probability of Bob's guessing one particular component in $\mathbf{x}$ is at most $\max(n_1^x, \ldots, n_{d_x}^x)/n$ and the probability of Alice's guessing one particular component in $\mathbf{y}$ is at most $\max(n_1^y, \ldots, n_{d_y}^y)/n$.

PROOF. Charlie receives $\boldsymbol{r}_x$ from Alice. But $\boldsymbol{r}_x$ can be obtained as the difference between $\hat{\boldsymbol{x}} + \boldsymbol{k}$ and $\boldsymbol{x} + \boldsymbol{k}$, where $\boldsymbol{k}$ is an n-vector with all its components set to k, where k is any real number. Hence, Charlie learns nothing about $\boldsymbol{x}$. A similar argument shows that Charlie learns nothing about $\boldsymbol{y}$.

On the other hand, Bob receives $\hat{\boldsymbol{x}}$ from Alice, which is a random permutation of $\boldsymbol{x}$. The number of different permutations of $\boldsymbol{x}$ is $\frac{n!}{n_1^x! n_2^x! \ldots n_{d_x}^x!}$; hence, the probability of Bob's guessing the correct one is 1 divided by this number. The argument for the probability of Alice's guessing $\boldsymbol{y}$ is analogous.

Finally, if Bob wants to guess a particular component of $\boldsymbol{x}$ given $\hat{\boldsymbol{x}}$, his best guess is the most frequent value in $\hat{\boldsymbol{x}}$, which is also the most frequent value in $\boldsymbol{x}$. The probability of the target component coinciding with the most frequent value is the relative frequency of the most frequent value. The argument when Alice wants to guess a component of $\boldsymbol{y}$ is analogous. □

If the probabilities given by Proposition 1 are not considered low enough (this may be the case if data are not diverse enough), then Protocol 1.1 should not be used. In this case, another option can be to combine permutation and noise addition into a hybrid of Protocol 1 and Protocol 1.1, as follows:

Protocol 1.2.

1. *Charlie sends Alice the seed for a common random generator of a random n-vector $\boldsymbol{r}_x$, and sends Bob the seed for a common random generator of a random n-vector $\boldsymbol{r}_y$ (or equivalently generates and sends the vectors if doing so is faster than Alice and Bob generating them).*

2. *Alice computes $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{r}_x$ and randomly permutes the values in $\hat{\boldsymbol{x}}$ to obtain $\hat{\boldsymbol{x}}' = \mathcal{P}_x(\hat{\boldsymbol{x}})$.*

3. *Alice sends $\hat{\boldsymbol{x}}'$ to Bob and $\boldsymbol{r}'_x = \hat{\boldsymbol{x}}' - \boldsymbol{x}$ to Charlie.*

4. *Bob computes $\hat{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{r}_y$ and randomly permutes the values in $\hat{\boldsymbol{y}}$ to obtain $\hat{\boldsymbol{y}}' = \mathcal{P}_y(\hat{\boldsymbol{y}})$.*

5. *Bob sends $\hat{\boldsymbol{y}}'$ to Alice and $\boldsymbol{r}'_y = \hat{\boldsymbol{y}}' - \boldsymbol{y}$ to Charlie.*

6. *Charlie sends $p = (\boldsymbol{r}'_x)^T \boldsymbol{r}'_y$ (note that p is a number) to CLARUS.*

7. *Bob sends* $t = (\hat{\boldsymbol{x}}')^T \boldsymbol{y}$ *to CLARUS.*

8. *Alice sends* $s_x = (\boldsymbol{r}'_x)^T \hat{\boldsymbol{y}}'$ *to CLARUS.*

9. *CLARUS computes*

$$t - s_x + p \ \ [= (\boldsymbol{x} + \boldsymbol{r}'_x)^T \boldsymbol{y} - (\boldsymbol{r}'_x)^T (\boldsymbol{y} + \boldsymbol{r}'_y) + (\boldsymbol{r}'_x)^T (\boldsymbol{r}'_y)] = \boldsymbol{x}^T \boldsymbol{y}.$$

In the above protocol, reusing the random vectors in successive instances is not needed, because the components are randomly permuted each time, so that an attacker cannot link the successive noise-added versions of the same original component of $\boldsymbol{x}$ (resp. $\boldsymbol{y}$). Regarding security, Protocol 1.2 is at least as secure as Protocol 1. Specifically:

Proposition 2 (Security). *Protocol 1.2 does not allow Charlie to learn* $\boldsymbol{x}$ *or* $\boldsymbol{y}$*, it does not allow Alice to learn* $\boldsymbol{y}$*, and it does not allow Bob to learn* $\boldsymbol{x}$*.*

Proof. Charlie receives $\boldsymbol{r}'_x$ from Alice. But $\boldsymbol{r}'_x$ can be obtained as the difference between $\hat{\boldsymbol{x}}' + \boldsymbol{k}$ and $\boldsymbol{x} + \boldsymbol{k}$, where $\boldsymbol{k}$ is an n-vector with all its components set to k, where k is any real number. Hence, Charlie learns nothing about $\boldsymbol{x}$. A similar argument shows that Charlie learns nothing about $\boldsymbol{y}$.

Regarding Alice and Bob, Protocol 1.2 clearly offers at least the same security as Protocol 1. In fact, due to the additional random permutation step and provided that there is some diversity in the original attribute values, it offers better security: it is free from the shortcomings of Protocol 1 related to poor choice of random vectors and to range inference. At best, the attacker can make inferences on permuted values. □

4.2. Secure scalar product with cryptography

Using cryptography to compute the scalar product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ privately owned by Alice and Bob, respectively, can be expected to increase the computational complexity with respect to non-cryptographic protocols. However, it is attractive in terms of security if it can be shown that for Alice to learn $\boldsymbol{y}$ or Bob to learn $\boldsymbol{x}$ they should break a cryptosystem that is known to be secure.

In [18], the authors proposed a cryptographic protocol based on the Paillier homomorphic cryptosystem [27]. Let $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_n)^T$. We remain under the honest-but-curious, non-sharing model: the CSPs neither deviate from the protocols nor pool the data fragments they hold. The protocol is depicted in Figure 2 and consists of the following steps:

Protocol 2.

Set-up phase:

1. *Alice generates a private and public key pair* (s_k, p_k) *and sends* p_k *to Bob.*

Scalar product:

2. *Alice generates the ciphertexts* $c_i = Enc_{p_k}(x_i; r_i)$*, where* r_i *is a random number in* $\mathbb{F}_N$*, for every* $i = 1, \ldots, n$*, and sends them to Bob.*

3. *Bob computes* $\omega = \prod_{i=1}^{n} c_i^{y_i}$.

4. *Bob generates a random plaintext* s_B*, a random number* r' *and sends* $\omega' = \omega Enc_{p_k}(-s_B; r')$ *to Alice.*

5. *Alice computes* $s_A = Dec_{s_k}(\omega') = \boldsymbol{x}^T\boldsymbol{y} - s_B$.

6. *Alice and Bob simultaneously exchange the values* s_A *and* s_B*, respectively, so that both can compute* $s_A + s_B = \boldsymbol{x}^T\boldsymbol{y}$.

Protocol 2 works in a finite field $\mathbb{F}_N$, where the order N must be large enough (as explained in Appendix A) and it is the product of two primes p and q of the same length and such that $\gcd(pq, (p-1)(q-1)) = 1$. In case Alice and Bob need to execute this protocol several times, they can reuse public and private keys and thus the set-up step (first step) needs to be executed only once. The number of computations required is: Bob must perform n exponentiations and one encryption, and Alice has to perform n encryptions and one decryption. Encryption involves computing two exponentiations and multiplying them, but one of the exponentiations can be precomputed. Decryption needs one exponentiation as its most expensive operation. The complexity of all these operations depends on N: the larger N, the more computationally demanding they are.

In Protocol 2, both Alice and Bob obtain the result. If we want only the proxy to learn it, we propose the following variation of the last steps:

Protocol 2.1.

4. *Bob generates a random plaintext* s_B*, a random number* r'*, sends* $\omega' = \omega Enc_{p_k}(-s_B; r')$ *to Alice and sends* s_B *to CLARUS.*

5. *Alice sends* $s_A = Dec_{s_k}(\omega') = \boldsymbol{x}^T\boldsymbol{y} - s_B$ *to CLARUS.*

6. *CLARUS computes* $s_A + s_B = \boldsymbol{x}^T\boldsymbol{y}$.

A sketch of Protocol 2.1 is given in Figure 4.

If it is possible for CLARUS to use an auxiliary cryptographic module, the number of computations can be reduced, and the protocol can be simplified as follows:

Protocol 2.2.

Set-up phase:

1. *CLARUS generates a private and public key pair* (s_k, p_k) *and sends* p_k *to Alice and Bob.*

Scalar product:

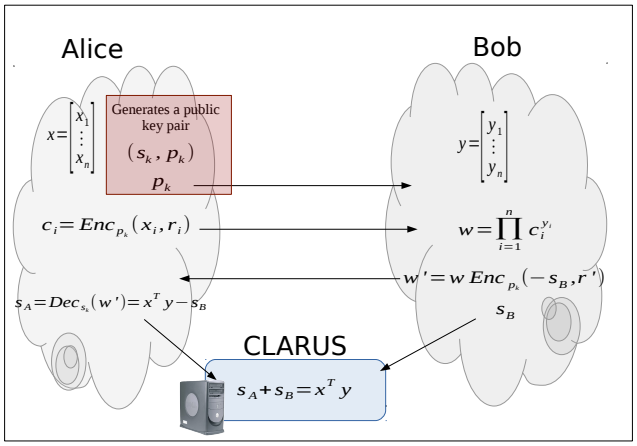

Figure 4: Protocol 2.1 is based on the Paillier homomorphic cryptosystem. In this case, the generation of the public key pair is left to Alice.

2. *Alice generates the ciphertexts* $c_i = Enc_{p_k}(x_i; r_i)$*, where* r_i *is a random number belonging to* $\mathbb{F}_N$*, for every* $i = 1, \ldots, n$ *and sends them to Bob.*

3. *Bob computes* $\omega = \prod_{i=1}^{n} c_i^{y_i}$ *and sends it to CLARUS.*

4. *CLARUS computes* $Dec_{s_k}(\omega) = \boldsymbol{x}^T \boldsymbol{y}$.

We have the following security result regarding the previous protocols.

Proposition 3 (Security). *If Paillier's cryptosystem is secure, then Protocols 2.1 and 2.2 are secure, in the sense that Alice cannot learn* $\boldsymbol{y}$ *and Bob cannot learn* $\boldsymbol{x}$.

Proof. It is proven in [18] that Protocol 2 is secure in the above sense if the Paillier cryptosystem is secure (see [27] about the security of this cryptosystem).

Regarding Protocol 2.1, the only modification with respect to Protocol 2 is that Alice and Bob do not share their results s_A and s_B, but they send these values to CLARUS. Since neither Alice nor Bob have more information than in Protocol 2, the security of Protocol 2 is preserved by Protocol 2.1.

Regarding Protocol 2.2, the only differences with Protocol 2 are that: Alice neither generates the key pair in the set-up phase nor decrypts ω' later; Bob does not generate and encrypt s_B; Alice and Bob do not share their results s_A and s_B, but they send these values to CLARUS. Neither Alice nor Bob have more information than in Protocol 2. Therefore, the security of Protocol 2 is preserved by Protocol 2.2. □

4.3. Example data analyses based on scalar products

In vertical splitting, analyses that involve only attributes in a single fragment are really fast and easy to compute: the cloud storing the fragment can compute and send the outputs of the analyses to the CLARUS proxy. Unfortunately, many statistical analyses, such as regression, classification, principal component analysis, etc., are likely to involve attributes stored in different fragments, and thus communication between clouds.

The sample correlation matrix $\hat{\boldsymbol{\zeta}}$ is fundamental for many statistical analyses. Let $\boldsymbol{X}$ be the original data set with n rows (records) and m columns (attributes $\boldsymbol{X}_1, \cdots, \boldsymbol{X}_m$). The sample correlation matrix of $\boldsymbol{X}$ can be computed as $\hat{\boldsymbol{\zeta}} = (\hat{\rho}_{ij})$ for $1 \leq i, j \leq m$, where

$$\hat{\rho}_{ij} = \frac{1}{n} \frac{\boldsymbol{X}_i^T \boldsymbol{X}_j - n\hat{\mu}_i \hat{\mu}_j}{\hat{\sigma}_i \hat{\sigma}_j} \tag{1}$$

with $\hat{\boldsymbol{\mu}}^T = (\hat{\mu}_1, \ldots, \hat{\mu}_m)$ being the vector of sample means and $\hat{\boldsymbol{\sigma}}^T = (\hat{\sigma}_1, \ldots, \hat{\sigma}_m)$ the vector of sample standard deviations of the attributes of $\boldsymbol{X}$. Regarding the scalar product $\boldsymbol{X}_i^T \boldsymbol{X}_j$ in the numerator of Expression (1), it can be viewed as a component of the following matrix

$$\begin{aligned} \boldsymbol{X}^T \boldsymbol{X} &= (\boldsymbol{X}_1 | \cdots | \boldsymbol{X}_m)^T (\boldsymbol{X}_1 | \cdots | \boldsymbol{X}_m) \\ &= \left(\begin{array}{c|c|c|c} \boldsymbol{X}_1^T \boldsymbol{X}_1 & \boldsymbol{X}_1^T \boldsymbol{X}_2 & \cdots & \boldsymbol{X}_1^T \boldsymbol{X}_m \\ \hline \boldsymbol{X}_2^T \boldsymbol{X}_1 & \boldsymbol{X}_2^T \boldsymbol{X}_2 & \cdots & \boldsymbol{X}_2^T \boldsymbol{X}_m \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{X}_m^T \boldsymbol{X}_1 & \boldsymbol{X}_m^T \boldsymbol{X}_2 & \cdots & \boldsymbol{X}_m^T \boldsymbol{X}_m \end{array} \right). \end{aligned}$$

Each element of $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\sigma}}$ and each diagonal element $\boldsymbol{X}_i^T \boldsymbol{X}_i$ can be separately computed by the respective cloud and sent to the CLARUS proxy, who calculates $\hat{\boldsymbol{\zeta}}$. Off-diagonal elements are also easy to compute if the involved attributes are stored in the same cloud. The most challenging task is therefore calculating the off-diagonal elements of $\boldsymbol{X}^T \boldsymbol{X}$ when the involved attributes are *not* in the same cloud. Note that it is not expensive for CLARUS to subsequently reassemble and store $\boldsymbol{X}^T \boldsymbol{X}$ and to derive $\hat{\boldsymbol{\zeta}}$ according to Expression (1): on the one hand, if $\boldsymbol{X}$ has n records and m attributes with $m \ll n$, $\boldsymbol{X}^T \boldsymbol{X}$ is an $m \times m$ matrix, and thus it is very small compared to the size of $\boldsymbol{X}$; on the other hand, the required computations to obtain $\hat{\boldsymbol{\zeta}}$ are simply operations between short vectors (of m components) or numbers.

Computing $\boldsymbol{X}_i^T \boldsymbol{X}_j$, for any $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ stored in different clouds, amounts to performing a secure scalar product of two vectors each held by a different party (where "secure" means without any party disclosing her vector to the other party). Therefore, obtaining the sample correlation matrix in vertical splitting among several clouds can be decomposed into several secure scalar products to be conducted between pairs of clouds.

Furthermore, being able to securely compute scalar products permits obtaining, in addition to sample correlation matrices, contingency tables (that is,

cross-tabulations) in a very simple way. To help computing a cross-tabulation cell between value x of attribute $\boldsymbol{X}_i$ and value x' of attribute $\boldsymbol{X}_j$, the cloud holding $\boldsymbol{X}_i$ computes an auxiliary binary attribute as follows:

$$\boldsymbol{aux}_{ix} = \begin{cases} 1 \text{ for records with } \boldsymbol{X}_i = x; \\ 0 \text{ for records with } \boldsymbol{X}_i \neq x; \end{cases}$$

similarly, the cloud holding $\boldsymbol{X}_j$ computes another auxiliary attribute:

$$\boldsymbol{aux}_{jx'} = \begin{cases} 1 \text{ for records with } \boldsymbol{X}_j = x'; \\ 0 \text{ for records with } \boldsymbol{X}_j \neq x'. \end{cases}$$

Finally, to count the joint occurrences of $\boldsymbol{X}_i = x$ and $\boldsymbol{X}_j = x'$, the clouds holding $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$, respectively, engage in a secure scalar product protocol of their attributes $\boldsymbol{aux}_{ix}$ and $\boldsymbol{aux}_{jx'}$. Note that this procedure is directly applicable to discrete numerical and categorical attributes, and can also be applied to continuous numerical attributes if discretized as intervals.

5. A sharing-resistant protocol in case of clouds without side knowledge

If the non-sharing assumption between clouds does not hold, then vertical partitioning alone cannot guarantee the privacy of the stored sensitive data. Here we show how the owner of a sensitive data set (represented by the CLARUS proxy in our case) can still use the computing power of several honest-but-curious, potentially sharing clouds to compute the matrix product $\boldsymbol{X}^T\boldsymbol{X}$ of her original data set $\boldsymbol{X}$ (which contains the scalar products of the columns of $\boldsymbol{X}$). With the new protocol we propose, no information sharing between the clouds can determine the original data set or its matrix product. The latter is only seen by the owner CLARUS. We assume here that the CSPs do not have any side knowledge about the original data set (such as its statistical structure).

Let $\boldsymbol{X}$ be an $n \times m$ matrix representing an original data set with n records and m attributes, where $n \gg m$. In this case we are interested in computing the matrix product $\boldsymbol{X}^T\boldsymbol{X}$. The protocol is depicted in Figure 5 and it runs in two phases: set-up and matrix product computation. The set-up phase needs to be run only once. The steps of the protocol are detailed next.

Protocol 3.
Set-up phase (data storage):

1. *CLARUS (the owner of $\boldsymbol{X}$) does:*
   (a) *Choose a random invertible $m \times m$ matrix $\boldsymbol{P}$.*
   (b) *Send $\boldsymbol{X}' = \boldsymbol{X}\boldsymbol{P}$ to t clouds $C_1, \ldots, C_t$.*
   (c) *Delete $\boldsymbol{X}'$ and $\boldsymbol{P}$.*
2. *For $i = 1$ to t, each cloud C_i does:*

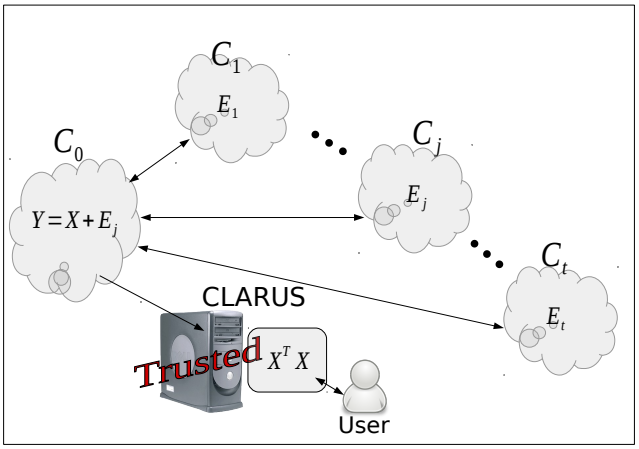

Figure 5: CSPs that share information but do not have side knowledge on the original data set. Despite sharing, the lack of knowledge of the CSPs on $\boldsymbol{X}$ prevents them from discovering with certainty which error matrix they ought to subtract from $\boldsymbol{Y}$ to get $\boldsymbol{X}$.

   (a) *Compute a random $n \times m$ matrix $\boldsymbol{E}_i$ such that it is orthogonal to $\boldsymbol{X}'$, that is, such that*

$$(\boldsymbol{X}')^T \boldsymbol{E}_i = (\boldsymbol{E}_i)^T \boldsymbol{X}' = \boldsymbol{0}.$$

   and send $\boldsymbol{E}_i$ to CLARUS.

   (b) *Compute $(\boldsymbol{E}_i)^T \boldsymbol{E}_i$ and send it to CLARUS.*

3. *CLARUS does:*

   (a) *Randomly select $j \in_R \{1, \ldots, t\}$.*

   (b) *Read $\boldsymbol{E}_j$ and $(\boldsymbol{E}_j)^T \boldsymbol{E}_j$, and discard all communications from clouds other than C_j.*

   (c) *Compute $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{E}_j$.*

   (d) *Store $\boldsymbol{Y}$ at cloud C_0.*

   (e) *Delete $\boldsymbol{X}, \boldsymbol{E}_j$ and $\boldsymbol{Y}$.*

Matrix product computation:

1. *Cloud C_0 computes $(\boldsymbol{Y})^T \boldsymbol{Y}$ and returns it to CLARUS.*

2. *CLARUS just subtracts*

$$\boldsymbol{Y}^T \boldsymbol{Y} - (\boldsymbol{E}_j)^T \boldsymbol{E}_j = \boldsymbol{X}^T \boldsymbol{X}. \tag{2}$$

We have the following correctness and security results.

Proposition 4 (Correctness). *Protocol 3 is correct.*

PROOF. If $\boldsymbol{P}$ is not invertible, then there exists at least one vector $\boldsymbol{b}$ in Ker($\boldsymbol{X}'$) such that $\boldsymbol{X}'\boldsymbol{b} = \boldsymbol{0}$ (but it is not necessary that $\boldsymbol{X}\boldsymbol{b} \neq \boldsymbol{0}$). We can always take $\boldsymbol{P}$ invertible without losing security (see Appendix B). The correctness of Equation (2) follows from

$$\begin{aligned}(\boldsymbol{Y})^T\boldsymbol{Y} &= (\boldsymbol{X})^T\boldsymbol{X} + (\boldsymbol{E}_j)^T\boldsymbol{E}_j + (\boldsymbol{X})^T\boldsymbol{E}_j + (\boldsymbol{E}_j)^T\boldsymbol{X} \\ &= (\boldsymbol{X})^T\boldsymbol{X} + (\boldsymbol{E}_j)^T\boldsymbol{E}_j,\end{aligned}$$

where in the last step we use orthogonality between $\boldsymbol{E}_j$ and $\boldsymbol{X}$, which in turn follows from orthogonality between $\boldsymbol{E}_j$ and $\boldsymbol{X}'$. □

In Section 5.2, we discuss how to create $\boldsymbol{E}_j$ so that $\boldsymbol{X}'$ preserves the means and the variance of $\boldsymbol{X}$.

Proposition 5 (Security). *If the CSPs are honest-but-curious and share information, but have no side knowledge on $\boldsymbol{X}$, Protocol 3 is secure in the sense that their probability of guessing the correct $\boldsymbol{X}$ and $\boldsymbol{X}^T\boldsymbol{X}$ is at most $1/t$.*

PROOF. Let us examine what the clouds receive. During the matrix product computation, none of the clouds receives any further information. They only receive information during the set-up.

In Step 1 of the set-up, clouds $C_1, \ldots, C_t$ receive matrix $\boldsymbol{X}'$, which is the original data set multiplied by a random matrix generated by CLARUS; hence $\boldsymbol{X}'$ leaks no information on $\mathbf{X}$ to the clouds.

In Step 3 of the set-up, cloud C_0 receives $\boldsymbol{Y}$, which is the original data set plus a random matrix $\boldsymbol{E}_j$ generated by one of the clouds. However, the sharing clouds cannot find out which of the noise matrices $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_t$ they have generated is the one that has been used to obtain $\mathbf{Y}$. The clouds could try an exhaustive method: subtract all the possible $\boldsymbol{E}_i$ from $\boldsymbol{Y}$, i.e. $\boldsymbol{S}_i = \boldsymbol{Y} - \boldsymbol{E}_i$, for $i = 1, \ldots, t$. Note that, since all $\boldsymbol{E}_i$ are random (we assume the clouds honestly follow the protocol), all t matrices $\boldsymbol{S}_i$ are different with overwhelming probability. But the clouds have no way to guess which $\boldsymbol{S}_i$ equals the original data set $\boldsymbol{X}$, because by assumption they neither know $\boldsymbol{X}$ nor have any side information on $\boldsymbol{X}$. Hence, even by sharing information, the clouds have no better strategy than randomly picking one of $S_1, \ldots, S_t$; the probability that they hit the correct $\boldsymbol{X}$ is $1/t$.

On the other hand, to obtain $\boldsymbol{X}^T\boldsymbol{X}$, Alice also needs to guess the correct $\boldsymbol{E}_j$ in Expression (2), which happens with probability $1/t$. □

5.1. Computing the attribute means and standard deviations

In addition to leveraging the clouds to compute $\boldsymbol{X}^T\boldsymbol{X}$, CLARUS can use them to compute the attribute means and standard deviations. Note that, according to Expression (1), the vector of means and the vector of standard deviations are needed to compute the sample correlation matrix of the data set. The following additions need to be done to Protocol 3 to compute the means:

- In Step 2b of Protocol 3 (set-up phase), each cloud C_i computes the vector of sums of the columns of $\boldsymbol{E}_i$, say $\boldsymbol{s}_i^T = (s_{i1}, \ldots, s_{im})$ and the vector of the squared sums of the columns of $\boldsymbol{E}_i$, that is, $\boldsymbol{ss}_i^T = ((s_{i1})^2, \ldots, (s_{im})^2)$; then C_i sends $\boldsymbol{s}_i^T$ and $\boldsymbol{ss}_i^T$ along with $\boldsymbol{E}_i$ and $(\boldsymbol{E}_i)^T\boldsymbol{E}_i$ to CLARUS.

- In Step 1 of Protocol 3 (matrix product computation), C_0 computes the vector of sums of the columns of $\boldsymbol{Y}$, say $\boldsymbol{s}_Y^T = (s_{Y1}, \ldots, s_{Ym})$, and the vector of squared sums of the columns of $\boldsymbol{Y}$, that is, $\boldsymbol{ss}_Y^T = ((s_{Y1})^2, \ldots, (s_{Ym})^2)$ and returns both vectors to CLARUS along with $(\boldsymbol{Y})^T\boldsymbol{Y}$.

- In Step 2 of Protocol 3 (matrix product computation), CLARUS computes the vector of sample means of $\boldsymbol{X}$, say $(\hat{\boldsymbol{\mu}}_X)^T = (\hat{\mu}_{X1}, \ldots, \hat{\mu}_{Xm})$ from the partial results obtained from the clouds as

$$\hat{\boldsymbol{\mu}}_X = \frac{\boldsymbol{s}_Y^T - \boldsymbol{s}_j^T}{n}.$$

  Additionally, CLARUS computes $\boldsymbol{ss}_X^T = ((s_{X1})^2, \ldots, (s_{Xm})^2)$ from the partial results obtained from the clouds as

$$\boldsymbol{ss}_X^T = \boldsymbol{ss}_Y^T - 2(s_{j1}s_{Y1}, \ldots, s_{jm}s_{Ym}) + \boldsymbol{ss}_j^T.$$

  Finally, CLARUS computes the vector of standard deviations of $\boldsymbol{X}$ as

$$\hat{\boldsymbol{\sigma}}_X^T = \left(\sqrt{\frac{(s_{X1})^2}{n} - (\mu_{X1})^2}, \ldots, \sqrt{\frac{(s_{Xm})^2}{n} - (\mu_{Xm})^2}\right).$$

The additional computations to be performed by CLARUS may seem substantial, but they are only $O(m)$ and, since $m \ll n$, they are much less than the additional $O(mn)$ computations performed by the clouds. On the other hand, the correctness of those additional computations follows from direct algebraic verification.

Security. The security of the extended version of Protocol 3 to compute the attribute means and standard deviations is the same as the security of the basic Protocol 3. Clearly, the proof of Proposition 5 also holds for the extended version of Protocol 3, because the computations and communications added in the extended version do not result in the clouds *receiving* any additional information (all additional communications are directed to CLARUS).

5.2. Preserving the attribute means in the masked data set

It may be desirable to preserve the means of attributes of $\boldsymbol{X}$ in the data set $\boldsymbol{Y}$ stored in the cloud. To that end, matrices $\boldsymbol{E}_i$, $i = 1, \ldots, t$ must be such that each of their columns adds to 0. We show how to obtain such matrices $\boldsymbol{E}_i$.

Given a random $n \times m$ matrix $\boldsymbol{B}_i$ such that $(\boldsymbol{X}')^T\boldsymbol{B}_i = (\boldsymbol{B}_i)^T\boldsymbol{X}' = \boldsymbol{0}$, take any two of the m column vectors of $\boldsymbol{B}_i$, say $\boldsymbol{b}_j$ and $\boldsymbol{b}_k$, such that their respective components do not add to zero. This may be infeasible in two cases:

- If all columns of $\boldsymbol{B}_i$ have their respective components adding to 0, then we can take $\boldsymbol{E}_i = \boldsymbol{B}_i$ and stop.
- If all columns of $\boldsymbol{B}_i$ except one have their components adding to 0, then we have a problem and we must choose a new random matrix $\boldsymbol{B}_i$.

Divide all components of $\boldsymbol{b}_j$ by the sum of the components, in order to obtain a vector $\hat{\boldsymbol{b}}_j$ such that its components add to 1; do the same to $\boldsymbol{b}_k$ and get $\hat{\boldsymbol{b}}_k$. By construction, it holds that $(\boldsymbol{X}')^T\hat{\boldsymbol{b}}_j = \boldsymbol{0}$, $(\hat{\boldsymbol{b}}_j)^T\boldsymbol{X}' = \boldsymbol{0}^T$, $(\boldsymbol{X}')^T\hat{\boldsymbol{b}}_k = \boldsymbol{0}$, $(\hat{\boldsymbol{b}}_k)^T\boldsymbol{X}' = \boldsymbol{0}^T$. Then build

$$\boldsymbol{E}_i = \boldsymbol{B}_i - s_1\boldsymbol{M}_1 - \ldots - s_m\boldsymbol{M}_m, \tag{3}$$

where $\boldsymbol{M}_l$ for $l \neq j$ is a matrix having all columns equal to $\boldsymbol{0}$ except the l-th column which is equal to $\hat{\boldsymbol{b}}_j$, $\boldsymbol{M}_j$ is a matrix having all columns equal to $\boldsymbol{0}$ except the j-th column which is equal to $\hat{\boldsymbol{b}}_k$, and, for $l = 1, \ldots, m$, s_l is the sum of the l-th column of $\boldsymbol{B}_i$. Clearly, each column of $\boldsymbol{E}_i$ adds to 0 and, by construction, $(\boldsymbol{X}')^T\boldsymbol{E}_i = (\boldsymbol{E}_i)^T\boldsymbol{X}' = \boldsymbol{0}$.

Security. The error matrices obtained with Expression (3) are still random and can be different from each other. On the other hand, these additional computations are done separately by each cloud. Thus, they require no exchange of information and hence do not affect the security of the basic protocol, stated in Proposition 5.

5.3. Preserving means and correlations in subdomains

One may wish to enable the computation of sample correlations in the cloud for subdomains of $\boldsymbol{X}$. Also, one may wish to preserve the means of attributes for records in the subdomain. For example, consider a medical data set, in which we define the following 20 subdomains: women aged 0 to 9, women aged 10 to 19, ..., women aged 90+, men aged 0 to 9, men aged 10 to 19, ..., men aged 90+.

In the example, one can split $\boldsymbol{X}$ into 20 data subsets $\boldsymbol{X}(1)$, ..., $\boldsymbol{X}(20)$. Then Protocol 3 (set-up phase) with the improvements described in Sections 5.1 and 5.2 is separately run for each $\boldsymbol{X}(i)$. It holds that the corresponding $\boldsymbol{Y}(i)$ preserves attribute means and Protocol 3 (matrix product computation) can be used to compute correlations within each $\boldsymbol{X}(i)$.

The price paid for considering subdomains is that it may no longer hold that $n_i \gg m$, where n_i is the number of records of $\boldsymbol{X}(i)$. This reduces the computational gain of using Protocol 3 and also the privacy of the $\boldsymbol{X}(i)$, because the number of degrees of freedom is reduced. However, as long as all n_i are substantially larger than m, considering subdomains is acceptable.

Security. We merely subdivide the problem into disjoint subproblems. In each subproblem, security is guaranteed as per Proposition 5.

5.4. Using a single cloud

If one assumes all clouds may share information, then from the security point of view the situation is equivalent to using a single cloud. We could then take $C_0 = C_1 = \cdots = C_t$ in Protocol 3. While security is not affected, there are performance pros and cons in using a single cloud:

- *Pros.* Using a single cloud is simpler and CLARUS saves communication at Step 1b, because $\boldsymbol{X}'$ only needs to be sent to one cloud. Also, one can modify the protocol at Step 1 for the cloud to compute $\boldsymbol{Y}^T\boldsymbol{Y} - (\boldsymbol{E}_i)^T\boldsymbol{E}_i$ for $i = 1, \ldots, t$, so that at Step 2, CLARUS only needs to pick $\boldsymbol{Y}^T\boldsymbol{Y} - (\boldsymbol{E}_j)^T\boldsymbol{E}_j$ as $\boldsymbol{X}^T\boldsymbol{X}$ without doing any computation. A similar modification could be done to the additional computations described in Section 5.1: the cloud could be asked to provide all candidate vectors of means and standard deviations under the t different error matrices, so that the only job left to CLARUS would be to pick the right vectors.

- *Cons.* When all the work needs to be done by a single cloud, the overall computation is likely to take longer (except if the cloud has a great computational power and/or is very efficient at parallelizing). Also, CLARUS can no longer discard any communication (as it did in Step 3b when communication coming from clouds other than C_j was discarded). Further, if we require the cloud to provide all $\boldsymbol{Y}^T\boldsymbol{Y} - (\boldsymbol{E}_i)^T\boldsymbol{E}_i$ for $i = 1, \ldots, t$, plus all candidate vectors of means and standard deviations under all error matrices, communication increases even more.

So, all in all, there is no clear advantage in using a single cloud: while CLARUS saves computation, it incurs more communication costs.

6. A sharing-resistant protocol robust against cloud side knowledge

In Protocol 3, if clouds have side knowledge on the statistical structure of $\boldsymbol{X}$ (for example, correlations between attributes, etc.), they can discard most of the "false" error matrices and focus on the (possibly unique) error matrix $\boldsymbol{E}_j$ such that $\boldsymbol{Y} - \boldsymbol{E}_j$ matches their side knowledge on $\boldsymbol{X}$. This would allow them to recover $\boldsymbol{X}$. We propose an alternative solution that addresses this issue but requires more set-up computation from CLARUS.

By the argument given in Section 5.4, a set of clouds sharing information is equivalent to a single cloud from a security point of view. Whereas Protocol 3 could work the same way with one or several clouds, in this section we present a new protocol designed for a single cloud. The proposed protocol is depicted in Figure 6 and its steps are detailed next.

Protocol 4.
Set-up phase (data storage):

CLARUS does:

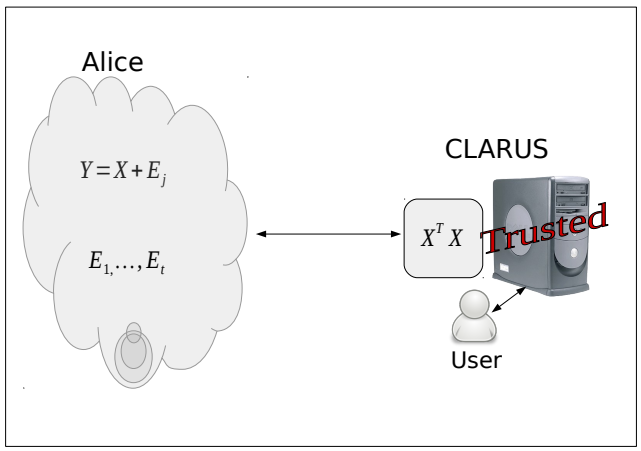

Figure 6: CSPs that share information and have side knowledge about the original data set are treated as if there was a single CSP. An anonymized version $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{E}_j$ of the original data set $\boldsymbol{X}$ is stored in the cloud, together with several plausible error matrices $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_t$ under the side information on $\boldsymbol{X}$ known to the CSP. The CSP does not know which error matrix should be subtracted from $\boldsymbol{Y}$ to recover $\boldsymbol{X}$.

1. *Anonymize the whole data set $\boldsymbol{X}$ using a randomized statistical disclosure control method to obtain a safe matrix $\boldsymbol{Y}$ and send $\boldsymbol{Y}$ to cloud Alice.*
2. *Randomly select $j \in_R \{1, \ldots, t\}$.*
3. *Let $\boldsymbol{E}_j = \boldsymbol{Y} - \boldsymbol{X}$ and let $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_{j-1}, \boldsymbol{E}_{j+1}, \ldots, \boldsymbol{E}_t$ be fake but plausible error matrices (in the sense that all $\boldsymbol{Y} - \boldsymbol{E}_i$ are plausible with the side information on $\boldsymbol{X}$ known to the clouds, for $i = 1, \ldots, t$).*
4. *Send $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_t$ to Alice.*
5. *Delete $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_t$.*

Matrix product computation:

1. *Alice computes $\boldsymbol{Y}^T\boldsymbol{Y}$.*
2. *For $i = 1, \ldots, t$, Alice computes:*

$$(\boldsymbol{Y} - \boldsymbol{E}_i)^T(\boldsymbol{Y} - \boldsymbol{E}_i) = \boldsymbol{Y}^T\boldsymbol{Y} + (\boldsymbol{E}_i)^T\boldsymbol{E}_i - \boldsymbol{Y}^T\boldsymbol{E}_i - (\boldsymbol{E}_i)^T\boldsymbol{Y}. \quad (4)$$

3. *For $i = 1, \ldots, t$, Alice sends $(\boldsymbol{Y} - \boldsymbol{E}_i)^T(\boldsymbol{Y} - \boldsymbol{E}_i)$ to CLARUS.*
4. *CLARUS picks $(\boldsymbol{Y} - \boldsymbol{E}_j)^T(\boldsymbol{Y} - \boldsymbol{E}_j)$ as $\boldsymbol{X}^T\boldsymbol{X}$.*

Note that, to compute the products in Step 2, the cloud needs to know $\boldsymbol{Y}$ and all error matrices. This is why this protocol only works for a single

cloud (or for clouds that share all information). The randomized statistical disclosure control (SDC) methods usable to obtain $\boldsymbol{Y}$ from $\boldsymbol{X}$ should be such that the matrix $\boldsymbol{Y} - \boldsymbol{X}$ looks random. Possible options include additive noise, multiplicative noise, synthetic data, etc. (see [19] for more details).

A key issue to Protocol 4 is how to obtain fake plausible error matrices $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_{j-1}, \boldsymbol{E}_{j+1}, \ldots, \boldsymbol{E}_t$. We propose to generate these fake error matrices as anonymized versions of the true $\boldsymbol{E}_j$. As above, the anonymization method should be such that the difference matrix between $\boldsymbol{E}_j$ and any anonymized version of it looks random. However, even if random-looking, the differences between $\boldsymbol{E}_j$ and its anonymized versions should be relatively small, so that the following two conditions are satisfied:

1. All the $\boldsymbol{E}_i$'s are similar enough to yield data sets $\boldsymbol{Y} - \boldsymbol{E}_i$ plausible under the side knowledge on $\boldsymbol{X}$ held by the clouds;

2. The matrices $\boldsymbol{Y} - \boldsymbol{E}_i$ for $i \neq j$ are not too similar to $\boldsymbol{Y} - \boldsymbol{E}_j = \boldsymbol{X}$.

Introducing random perturbations that achieve a good trade-off between protection and preservation of the structure of data is precisely the purpose of the aforementioned SDC methods. Note that, whereas when anonymizing $\boldsymbol{X}$ into $\boldsymbol{Y}$ the only purpose was protection (and perturbations could be large), when anonymizing $\boldsymbol{E}_j$ into $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_{j-1}, \boldsymbol{E}_{j+1}, \ldots, \boldsymbol{E}_t$, parameters for the SDC methods yielding smaller perturbations must be chosen to attain the previously mentioned protection/preservation trade-off. We can now state the following security result:

Proposition 6 (Security). *If $\boldsymbol{Y} - \boldsymbol{X}$ looks random, all $\boldsymbol{Y} - \boldsymbol{E}_i$ for $i = 1, \ldots, t$ are plausible under the CSP's side knowledge on $\boldsymbol{X}$, and $\boldsymbol{Y} - \boldsymbol{E}_i$ for $i = 1, \ldots, t$, $i \neq j$ are not too similar to $\boldsymbol{Y} - \boldsymbol{E}_j = \boldsymbol{X}$, then Protocol 4 is secure in the sense that the probability of the CSP guessing the correct $\boldsymbol{X}$ and $\boldsymbol{X}^T\boldsymbol{X}$ is at most $1/t$.*

Proof. Let us examine what the cloud Alice receives. During the matrix product computation, Alice receives no further information. She only receives information during the set-up.

In Step 1 of the set-up, Alice receives an anonymized version $\boldsymbol{Y}$ of the original data set $\boldsymbol{X}$. By assumption, $\boldsymbol{Y} - \boldsymbol{X}$ looks random, so $\boldsymbol{Y}$ does not leak $\boldsymbol{X}$.

Finally, in Step 4 of the set-up, Alice receives error matrices $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_t$. By assumption, Alice's side knowledge does not allow her to single out $\boldsymbol{Y} - \boldsymbol{E}_j$ (which is equal to $\boldsymbol{X}$) from $\boldsymbol{Y} - \boldsymbol{E}_i$ for $i = 1, \ldots, t$. On the other hand, also by assumption, $\boldsymbol{Y} - \boldsymbol{E}_i$ for $i = 1, \ldots, t$, $i \neq j$ are not too similar to $\boldsymbol{Y} - \boldsymbol{E}_j$.

Hence, Alice has no better strategy than randomly picking one of $\boldsymbol{Y} - \boldsymbol{E}_i$ for $i = 1, \ldots, t$ as $\boldsymbol{X}$; her probability of hitting the correct $\boldsymbol{X}$ is $1/t$.

On the other hand, to obtain $\boldsymbol{X}^T\boldsymbol{X}$, Alice also needs to guess the correct $\boldsymbol{E}_j$, which happens with probability $1/t$. □

6.1. Computing the attribute means and standard deviations

If attribute means and standard deviations are to be computed (for example, for CLARUS to obtain the sample correlation matrix of the data set), then some computations need to be added to Protocol 4:

- In Step 2 of Protocol 4 (matrix product computation), for $i = 1, \ldots, t$, Alice computes in an analogous way as in Section 5.1:
  1. $\boldsymbol{s}_Y^T$ and $\boldsymbol{ss}_Y^T$ corresponding to $\boldsymbol{Y}$;
  2. $\boldsymbol{s}_i^T$ and $\boldsymbol{ss}_i^T$ corresponding to $\boldsymbol{E}_i$;
  3. $\hat{\boldsymbol{\mu}}_{Y-E_i} = (\boldsymbol{s}_Y^T - \boldsymbol{s}_i^T)/n$;
  4. $\boldsymbol{ss}_{Y-E_i} = ((s_{Y-E_i,1})^2, \ldots, (s_{Y-E_i,m})^2)$ as
  $$\boldsymbol{ss}_{Y-E_i} = \boldsymbol{ss}_Y^T - 2(s_{i1}s_{Y1}, \ldots, s_{im}s_{Ym}) + \boldsymbol{ss}_i^T.$$
  5. The vector of standard deviations of $\boldsymbol{Y} - \boldsymbol{E}_i$:
  $$\hat{\boldsymbol{\sigma}}_{Y-E_i}^T = \left(\sqrt{\frac{(s_{Y-E_i,1})^2}{n} - (\mu_{Y-E_i,1})^2}, \ldots, \sqrt{\frac{(s_{Y-E_i,m})^2}{n} - (\mu_{Y-E_i,m})^2} \right).$$
- In Step 3 of Protocol 4, Alice sends $\hat{\boldsymbol{\mu}}_{Y-E_i}$ and $\hat{\boldsymbol{\sigma}}_{Y-E_i}$ to CLARUS, for $i = 1, \ldots, t$.
- In Step 4 of Protocol 4, CLARUS picks $\hat{\boldsymbol{\mu}}_{Y-E_j}$ as $\hat{\boldsymbol{\mu}}_X$ and $\hat{\boldsymbol{\sigma}}_{Y-E_j}$ as $\hat{\boldsymbol{\sigma}}_X$.

Security. The security of the extended version of Protocol 4 to compute the attribute means and standard deviations is the same as the security of the basic Protocol 4. The proof of Proposition 6 also holds for the extended version of Protocol 4, because the computations and communications added in the extended version do not result in Alice *receiving* any additional information.

6.2. Preserving means and other statistics

In Section 5.2 above, we discussed how the data set $\boldsymbol{Y}$ stored in the cloud under Protocol 3 could exactly preserve the attribute means of $\boldsymbol{X}$. This is even easier for Protocol 4, where it would be sufficient for all columns of all error matrices $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_t$ to add to zero. Given $\boldsymbol{E}_j$ whose columns add to zero, it is easy to generate anonymized versions of it (the other error matrices $\boldsymbol{E}_1, \ldots, \boldsymbol{E}_{j-1}, \boldsymbol{E}_{j+1}, \ldots, \boldsymbol{E}_t$) whose columns also add to zero (see [19]).

Furthermore, we can do more than preserving means. In the discussion above, we have not required the anonymized $\boldsymbol{Y}$ to preserve the statistical properties of $\boldsymbol{X}$. If the parameters of the SDC method used to transform $\boldsymbol{X}$ into $\boldsymbol{Y}$ are carefully chosen, many statistical properties of $\boldsymbol{X}$ can be exactly or approximately preserved by $\boldsymbol{Y}$. The specific preserved properties and whether preservation is only approximate depend on the particular SDC method used (see [19]).

7. Comparison among methods

We compare here the protocols described in the previous sections against benchmark solutions that consist of CLARUS downloading the data from the cloud or clouds and computing locally.

Specifically, the protocols to compute scalar products over vertically split data described in Section 4 are compared to the following benchmark protocol:

Protocol 5.

1. *Alice and Bob send* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *to CLARUS, respectively.*
2. *CLARUS locally computes* $\boldsymbol{x}^T\boldsymbol{y}$.

If we relax the non-sharing assumption, as in Sections 5 and 6, then we need a different benchmark which no longer relies on data splitting (note that Protocol 5 splits data among Alice and Bob). We will use as such alternative benchmark the storage of the whole data set $\boldsymbol{X}$ (containing in particular vectors $\boldsymbol{x}$ and $\boldsymbol{y}$) in a single cloud in encrypted form. In this setting, CLARUS first encrypts the data set, stores it in the cloud and, when it needs to compute a scalar product, it downloads and decrypts the data set, as detailed in the following protocol:

Protocol 6.

Set-up phase:

1. *CLARUS encrypts the original data set* $\boldsymbol{E} = Enc(\boldsymbol{X})$
2. *CLARUS sends* $\boldsymbol{E}$ *to a cloud Alice for remote storage, and deletes* $\boldsymbol{X}$ *from local storage.*

Matrix product computation:

3. *CLARUS requests* $\boldsymbol{E}$ *from Alice, decrypts* $\boldsymbol{X} = Dec(\boldsymbol{E})$ *and performs the computation* $(\boldsymbol{X})^T\boldsymbol{X}$.

Encryption and decryption can be performed using a fast symmetric cryptosystem, such as the Advanced Encryption Standard (AES), which takes time linear in the number of records/vector components n, as well as ciphertexts similar in size to the corresponding plaintexts.

We now evaluate the computational cost for Alice, Bob, CLARUS and the total computation under each protocol. Moreover, operations that do not need to be repeated each time the protocol is executed, specifically the generation of cryptographic keys in Protocols 2.1 and 2.2, are separately counted as set-up costs.

Assuming that the clouds have unlimited storage, it is reasonable to assume that they can store any random matrices or vectors that may need to be reused. In contrast, we do not assume unlimited storage at CLARUS; therefore, we assume the proxy just stores the random seeds and (re)generates random matrices

Table 1: **(Set-up computation and communication costs of Protocols 1, 1.1, 1.2, 2.1, 2.2, 5 and 6)**: n is the length of the private vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. γ represents the maximum length of the numbers in the vectors and matrices used in the protocols. N is the size of the plaintext field used by Paillier (see Appendix A). The computation cost is presented in terms of the costliest operations performed in each case. Protocols 2.1 and 2.2 have two different computation costs depending on the choice of N (they are separated by "|" in the table): if the smallest possible N is taken, then the private vectors need to be read (see Appendix A for details). The communication cost is the exact amount of data transmitted. The Paillier key generation cost is indicated with "PKgen", the AES keys generation with "RNDgen" and the AES encryption with "AESencr". Note: in protocols not requiring the presence of Bob or Charlie, their costs are indicated with "−".

	Set-up							
	Computation				Communication			
	Alice	Bob	Charlie	CLARUS	Alice	Bob	Charlie	CLARUS
Prot. 1	0	0	0	0	0	0	0	0
Prot. 1.1	0	0	0	0	0	0	0	0
Prot. 1.2	0	0	0	0	0	0	0	0
Prot. 2.1	n read + PKgen | PKgen	n read|0	–	0	$3\log_2 N$	$3\log_2 N$	–	0
Prot. 2.2	n read|0	n read|0	–	PKgen	$3\log_2 N$	$3\log_2 N$	–	$6\log_2 N$
Prot. 5	0	0	–	0	0	0	–	0
Prot. 6	0	–	–	n AESencr +1 RNDgen	$2n\gamma$	–	–	$2n\gamma$

or vectors when needed. Also, the cost of a communication is associated both to the sender (who needs to send the data) and to the receiver (who needs to read the data). We use a parameter γ to represent the maximum length of the numbers in the vectors and matrices used in the protocols. For the case of Protocols 2.1 and 2.2, lengths are a function of the size N of the field used by the Paillier cryptosystem (see Appendix A): the public key is $3\log_2 N$ bits long, the secret key is $\log_2 N$ bits long, ciphertexts are $2\log_2 N$ bits long and plaintexts are $\log_2 N$ bits long. We consider that, whenever possible, the participants send the seeds of random vectors and matrices, rather than the vectors and matrices themselves. If communications are very fast and/or vectors are very short, sending the vectors rather than the seeds might be preferable (see related experimental results in Section 8).

7.1. Comparison for scalar products over split data

In this section, we compare the protocols based on data splitting (Protocols 1, 1.1, 1.2, 2.1 and 2.2) with the benchmark Protocols 5 and 6. We do not detail the costs of Protocol 2, because they are mostly equivalent to those of the two variants, Protocols 2.1 and 2.2.

Table 1 shows the set-up computation costs and set-up communication costs incurred by all protocols. For the computation cost, just giving the order of magnitude of the complexity is not accurate enough (e.g. n additions are faster than n multiplications, even if we have $O(n)$ computation in both cases); therefore we give the complexity in terms of the costliest operation performed in each case. For instance, "read" means reading the vector, "AESencr" means AES encryption of the vectors and "RNDgen" is one random number generation. When the stored data are updated (that is, records are added, changed or removed), the set-up phase (key generation) of Protocols 2.1 and 2.2 needs to be repeated

Table 2: **(Long-term and temporary storage costs of Protocols 1, 1.1, 1.2, 2.1, 2.2, 5 and 6)**: n is the length of the private vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; γ represents the maximum length of the numbers in the vectors and matrices used in the protocols. N is the size of the plaintext field used by Paillier (see Appendix A). Note: in protocols not requiring the presence of Bob or Charlie, their costs are indicated with "–".

	Storage							
	Long-term				Temporary			
	Alice	Bob	Charlie	CLARUS	Alice	Bob	Charlie	CLARUS
Prot. 1	$(2n+1)\gamma$	$(2n+1)\gamma$	0	0	$(2n+1)\gamma$	$(2n+1)\gamma$	$(2n+3)\gamma$	4γ
Prot. 1.1	$n\gamma$	$n\gamma$	0	0	$(2n+1)\gamma$	$(2n+1)\gamma$	$(2n+1)\gamma$	4γ
Prot. 1.2	$(2n+1)\gamma$	$(2n+1)\gamma$	0	0	$(4n+1)\gamma$	$(4n+1)\gamma$	$(2n+3)\gamma$	4γ
Prot. 2.1	$n\gamma+$ $4\log_2 N$	$n\gamma$	–	0	$2\gamma+$ $2(n+1)\log_2 N$	$3\gamma+$ $2(n+2)\log_2 N$	–	3γ
Prot. 2.2	$n\gamma$	$n\gamma$	–	$4\log_2 N$	$\gamma+$ $2(n+1)\log_2 N$	$2(n+1)\log_2 N$	–	$\gamma+2\log_2 N$
Prot. 5	$n\gamma$	$n\gamma$	–	0	0	0	–	$(2n+1)\gamma$
Prot. 6	$2n\gamma$	–	–	0	0	–	–	$(4n+1)\gamma$

only if some of the new values are greater than the order N of the finite field in use; otherwise, it is possible to reuse the same keys without losing security. Protocol 6 requires downloading, decrypting, updating and re-encrypting all the records.

If in Protocols 2.1 and 2.2 one wants to save storage by using the smallest possible N that does not result in overflow, it takes $2n$ "read" effort (to check all vector elements). If one just takes a large N, say a 1024-bit N, then the key generation cost is constant. This is why the n reads have been marked as optional in Table 1. See Appendix A for details. At most, the key generation for both Protocol 2.1 and Protocol 2.2 requires 3 random generations, 1 modular multiplicative inversion and 1 least common multiple computation, which is represented with "PKgen" in the table. It is possible to use the same keys also for different data sets belonging to the same field $\mathbb{F}_N$; therefore, the update of elements in the private vectors does not require generating new keys. Protocol 6 also requires a set-up phase, that is, the key generation for the AES cryptosystem and the encryption of the private vectors. Compared to Protocols 2.1 and 2.2, the set-up phase of Protocol 6 needs to be repeated every time that the private vectors are changed.

Table 1 also shows the communication cost of the set-up phase. As said above, the cost of communicating a number of bits is incurred by *both* the sender (who must write them to the channel) and the receiver (who must read them from the channel). Only Protocol 2.1, Protocol 2.2 and Protocol 6 involve a set-up communication cost, due to the exchange of the public key for the two former protocols and the transmittal of the encrypted private vectors for the latter one.

Table 2 shows the long-term and temporary data storage costs (temporary storage is the one needed only to conduct a certain calculation at some point). In Protocol 1, Alice needs long-term storage for her data vector $\boldsymbol{x}$ and also for the random vector $\boldsymbol{r}_x$, which is needed for potential reuse; similarly for Bob

Table 3: **(Execution costs for Protocols 1, 1.1, 1.2, 2.1, 2.2, 5 and 6)**: n is the length of the private vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; γ represents the maximum length of the numbers in the vectors and matrices used in the protocols. Finally, Charlie only appears in Protocols 1, 1.1 and 1.2, and Bob does not appear in Protocol 6; we indicate the absence of a cloud with "–". The computation cost is presented in terms of the costliest operations performed in each case; the communication cost is the exact amount of data transmitted. In Protocol 1 we have considered the most usual case in which there is no reuse; for each reused private vector, $n\gamma$ communication cost is shifted from Charlie to the reusing cloud, and the computational cost for Charlie decreases by n RNDgen.

	Computational cost				Communication cost				Who needs a crypt. module
	Alice	Bob	CLARUS	Charlie	Alice	Bob	CLARUS	Charlie	
Prot. 1	n prod. n RNDgen.	n prod. n RNDgen.	2 sum.	$2n$ RNDgen. $+n$ prod.	$(2n+2)\gamma$	$(2n+2)\gamma$	3γ	3γ	none
Prot. 1.1	n prod. n RNDgen.	n prod. n RNDgen.	2 sum.	n prod.	$(3n+1)\gamma$	$(3n+1)\gamma$	3γ	$(2n+1)\gamma$	none
Prot. 1.2	n prod. n RNDgen.	n prod. n RNDgen.	2 sum.	n prod.	$(3n+2)\gamma$	$(3n+2)\gamma$	3γ	$(2n+3)\gamma$	none
Prot. 2.1	n RNDgen. $+n$ encr.+1 decr.	n prod. 2 RNDgen +1 encr.	1 sum.	–	$2(n+1)\log_2 N$ $+\gamma$	$2(n+1)\log_2 N$ $+\gamma$	2γ	–	Alice
Prot. 2.2	n RNDgen. $+n$ encr.	n prod.	1 decr.	–	$2n\log_2 N$	$(2(n+1)\log_2 N$	$2\log_2 N$	–	CLARUS
Prot. 5	0	0	n prod.	–	$n\gamma$	$n\gamma$	$2n\gamma$	–	none
Prot. 6	0	–	n prod. + n AESdecr.	–	$2n\gamma$	–	$2n\gamma$	–	CLARUS

regarding $\boldsymbol{y}$ and $\boldsymbol{r}_y$. In Protocols 1.1 and 1.2, the random vectors do not need to be reused, so less long-term storage is needed by Alice and Bob; on the other hand, computing the random permutations in those protocols takes just n random number generations and no auxiliary storage, by using Durstenfeld's algorithm [15]. Only the benchmark Protocols 5 and 6 require CLARUS to (temporarily) store a large amount of data, namely the downloaded data, plus the decrypted data if the downloaded data are encrypted.

Table 3 shows the computational and communication costs incurred by the execution of the above mentioned protocols (after set-up). As said above, in Protocols 1.1 and 1.2 a permutation of n elements takes n random number generations. Protocols 1, 1.1, 1.2 and 2.1 have all similar costs for CLARUS. Protocol 2.2 requires more computation and communication from CLARUS, but the difference can be reduced if CLARUS has a cryptographic module. It is worth noting that *all protocols have constant computation, storage and communication costs for CLARUS; hence, they clearly outperform the benchmark Protocols 5 and 6, which require from CLARUS computation, storage and communication that increase linearly with the data set size n.*

If we count costs for all parties involved, Protocol 1 is the most efficient one, closely followed by Protocol 1.1 and Protocol 1.2 (note that these three protocols do not require any set-up). If we focus on the cost/security trade-off, Protocol 1.2 is probably the best choice.

7.2. Comparison for the sharing-resistant alternative without side knowledge

Table 4 shows all the costs for Protocols 3 and 6. Alice represents C_0, the cloud storing the anonymized data set $\boldsymbol{Y}$. The other clouds $C_1, \ldots, C_m$

Table 4: **(Costs for Protocols 3 and 6)**: $\boldsymbol{X}$ is the $n \times m$ matrix (with $n \gg m$) containing the original data set. γ represents the maximum length of the numbers in the vectors and matrices, and t is the number of clouds involved in the protocol.

	Protocol 3			Protocol 6	
	Alice (C_0)	Bob (C_j)	CLARUS	Alice	CLARUS
Set-up computation	0	$3nm^2$ prod.	nm^2 prod. $+m^2$ RNDgen.	0	nm AESencr.
Set-up communication	nm	$(2nm + m^2)\gamma$	$((t+2)nm + m^2)\gamma$	$nm\gamma$	$nm\gamma$
Long-term storage	$nm\gamma$	$nm\gamma$	$m^2\gamma$	$nm\gamma$	$nm\gamma$
Temporary storage	0	0	$2m^2\gamma$	0	$(nm + m^2)\gamma$
Matrix product computation	nm^2 prod.	0	m^2 subtr.	0	nm AESdecr. $+nm^2$ prod.
Matrix product communication	$m^2\gamma$	0	$m^2\gamma$	$nm\gamma$	$nm\gamma$

perform similar amounts of computation and so we represent all of them by Bob. Protocol 3 needs a set-up phase, in which:

- CLARUS needs to compute the random invertible $m \times m$ matrix $\boldsymbol{P}$ (as described in Appendix B). Then it must multiply $\boldsymbol{X}$ times $\boldsymbol{P}$ and add $\boldsymbol{E}_j$ to $\boldsymbol{X}$.
- Each cloud needs to compute the orthogonal complement of $\boldsymbol{Y}$ (as described in Appendix C), which takes $2nm^2$ products (using the Gram-Schmidt algorithm for QR decomposition), plus $(\boldsymbol{E}_j)^T\boldsymbol{E}_j$, which takes nm^2 products.

The set-up phase is performed just once, unless $\boldsymbol{X}$ is modified (in which case it needs to be repeated).

Regarding storage, in Protocol 3 each cloud stores one $n \times m$ matrix ($\boldsymbol{X'}$ for $C_1, \ldots, C_t$, and $\boldsymbol{Y}$ for C_0), whereas CLARUS stores three $m \times m$ matrices ($(\boldsymbol{Y})^T\boldsymbol{Y}$ and $(\boldsymbol{X})^T\boldsymbol{X}$ as temporary storage and $(\boldsymbol{E}_j)^T\boldsymbol{E}_j$ as long-term storage). As to communication, at set-up CLARUS sends one $n \times m$ matrix to clouds $C_1, \ldots, C_t$ (this can be done in a single message if using broadcast or if using a single cloud to do all computations) and one $n \times m$ matrix $\boldsymbol{Y}$ to C_0; each of $C_1, \ldots, C_t$ returns one $n \times m$ matrix and one $m \times m$ matrix to CLARUS; C_0 returns one $m \times m$ matrix to CLARUS.

To compute the matrix product (after set-up), in Protocol 3 C_0 needs to compute $(\boldsymbol{Y})^T\boldsymbol{Y}$, which takes nm^2 products. CLARUS only needs to compute m^2 subtractions. As to communications during matrix product, only C_0 needs to send an $m \times m$ matrix, which makes $m^2\gamma$ bits.

With the benchmark Protocol 6, CLARUS needs less set-up communication than with Protocol 3, but more communication during the matrix product computation. The difference between the two protocols regarding storage is substantial: Protocol 3 requires CLARUS to use much less temporary and long-term storage. Another important difference refers to the computing time for the matrix product: in the benchmark protocol, CLARUS needs to perform nm

Table 5: **(Costs for Protocol 4)**: γ represents the maximum length of the numbers in the vectors and matrices. "anon" stands for the anonymization cost of a value and "read" stands for the cost of reading a value.

	Protocol 4		Protocol 6	
	Alice	CLARUS	Alice	CLARUS
Set-up computation	0	$(t+1)nm$ anon.	0	nm AESencr.
Set-up communication	$(t+1)nm\gamma$	$(t+1)nm\gamma$	$nm\gamma$	$nm\gamma$
Long-term storage	$(t+1)nm\gamma$	γ	$nm\gamma$	$nm\gamma$
Temporary storage	$(3t+1)m^2$	$m^2\gamma$	0	$(nm+m^2)\gamma$
Matrix product computation	$(2t+1)nm^2$ prod. $+tm^2$ reads	0	0	nm AESdecr. $+nm^2$ prod.
Matrix product communication	$(3t+1)m^2\gamma$	$(3t+1)m^2\gamma$	$nm\gamma$	$nm\gamma$

decryptions (to decrypt the entire data set) plus nm^2 products, which is clearly more work than the m^2 subtractions needed under Protocol 3.

In summary, the advantage of Protocol 3 is that it permits different statistical analyses without requiring the use of the original data set $\boldsymbol{X}$: just using $\boldsymbol{Y}$ suffices and most of the computational burden falls on the C_0 cloud. In contrast, Protocol 6 requires CLARUS to download and decrypt $\boldsymbol{X}$ before performing any analysis.

7.3. Comparison for the sharing-resistant alternative with side knowledge

Table 5 shows all the costs for Protocols 4 and 6. Protocol 4 requires a set-up phase to anonymize the original data set, compute the true error matrix and obtain the t plausible fake error matrices by means of an SDC method.

To compute Expression (4) t times, Alice first computes $\boldsymbol{Y}^T\boldsymbol{Y}$ once, and then, for $i = 1, \ldots, t$, she computes $(\boldsymbol{E}_i)^T\boldsymbol{E}_i$, $\boldsymbol{Y}^T\boldsymbol{E}_i$ and $(\boldsymbol{E}_i)^T\boldsymbol{Y}$. This takes $(2t+1)nm^2$ products, and transposing $(\boldsymbol{Y})^T\boldsymbol{E}_i$ into $(\boldsymbol{E}_i)^T\boldsymbol{Y}$ for $i = 1, \ldots, t$ takes tm^2 reads.

When comparing Protocol 4 with Protocol 6, we can see that the latter has smaller set-up costs for CLARUS, but larger costs for the matrix computation phase. Although the set-up costs are higher for Protocol 4, they allow usefully releasing the data protected with anonymization to potential users (such as researchers), which is not possible with Protocol 6 (because protected data are encrypted in the latter protocol).

8. Experimental results

This section details the experimental results obtained by implementing the proposed protocols in Java in a multi-cloud scenario. Since Protocols 1 and 2 had security and functionality issues that motivated Protocols 1.1, 1.2, 2.1, and 2.2, we focused on implementing the latter protocols.

The reported experiments were conducted using the first two attributes of the California housing data set (CADATA, [4]), a usual test data set in the statistical disclosure control literature that contains 9 numerical attributes and $20,640$ records. Let $\mathbf{x}$ and $\mathbf{y}$ represent the two selected attributes of CADATA, and $\mathbf{X}$ be the matrix containing $\mathbf{x}$ and $\mathbf{y}$.

Table 6: **(Computational specifications of CLARUS and CSPs)** CLARUS, the trusted proxy, was run on a local computer. CERSEI is a local server mimicking a for-payment CSP. AWS represents a free-of-charge t2.micro Amazon EC2 instance.

Machine	Operating System	Width(bits)	CPU(GHz)	RAM(GB)	HDD(GB)
CLARUS	Windows 7	64	2.5	8	500
CERSEI	Ubuntu 14.4 LTS	64	3.4	16	500
AWS	Ubuntu Server 16.04 LTS	64	2.4	1	30

First, we ran the tests on Amazon Web Services (AWS), a public CSP that offers 12 months free tier. The tests were performed on a t2.micro Amazon EC2 instance for each cloud. Since the computing power and main storage were substantially capped in this free-of-charge service, and communication with it was slow, we took it as representing the low-end scenario a user can expect from a CSP. Second, we also used a local server (CERSEI) offering more computational power and main storage, as well as faster communication, in order to mimic the service that can be expected from a for-payment CSP. In both cases, we used another local computer to run CLARUS on it, that worked as the proxy located in a trusted domain. Table 6 summarizes the specifications of CLARUS, CERSEI and the AWS instance.

The computational cost was measured as the time in seconds that each machine spent to perform the computations specified by the protocols. The communication cost was measured as an approximation of the time each cloud spent at sending data (writing to the channel) or receiving data (reading from the channel). As explained in Section 7, we consider in general that the participants send the seeds of random vectors and matrices, rather than the vectors and matrices themselves. The reason is that generating a random vector takes normally less time than sending it. However, the communication time depends on many factors (user internet connection, distance between users, network load, etc.). In our experimental setting, we compared the times for generating and sending vectors of several sizes n; the results are reported in Table 7, where it can be seen that sending a random vector takes longer than generating it at the recipient for sizes $n > 10^4$.

Table 7: **(Time to send a random vector vs time to generate it at the recipient)**: n is the number of elements of a random vector $\mathbf{x}$ and the times are given in seconds. Sending is faster only for short vectors.

Time (s) \ n	10^3	10^4	10^5	10^6
to generate $\mathbf{x}$	2.08×10^{-3}	4.6×10^{-3}	8.4×10^{-3}	2.07×10^{-2}
to send $\mathbf{x}$	2.8×10^{-4}	3.5×10^{-3}	1.9×10^{-2}	1.19×10^{-1}

Table 8: **(Long-term and temporary storage costs of Protocols 1.1, 1.2, 2.1, 5 and 6)**: Private vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ have length $n = 20,640$. The representation of the numbers takes $\gamma = 8$ bytes. Elements in the field $\mathbb{F}_N$ used by the Paillier cryptosystem have 65 bytes. We indicate the absence of a cloud with "–". The values are given in bytes.

	Storage (B)							
	Long-term				Temporary			
	Alice	Bob	Charlie	CLARUS	Alice	Bob	Charlie	CLARUS
Prot. 1.1	1.7×10^5	1.7×10^5	0	0	3.3×10^5	3.3×10^5	3.3×10^5	32
Prot. 1.2	3.3×10^5	3.3×10^5	0	0	6.6×10^5	6.6×10^5	3.3×10^5	32
Prot. 2.1	1.7×10^5	1.7×10^5	–	0	2.5×10^5	2.5×10^5	–	24
Prot. 5	1.7×10^5	1.7×10^5	–	0	0	0	–	3.3×10^5
Prot. 6	3.3×10^5	–	–	0	0	–	–	6.6×10^5

8.1. Experimental results for scalar products over split data

In this section, we detail storage, computation and communication costs of Protocols 1.1, 1.2 and 2.1 (based on data splitting) and the benchmark Protocols 5 and 6. We do not detail the costs of Protocol 2.2, because they are basically equivalent to those of Protocol 2.1.

Table 8 shows the comparison of long-term and temporary storage measured in bytes. The storage does not depend on the particular CSP used. Long-term storage turns out to be similar in all protocols for all players involved; the only remarkable difference occurs in Protocol 6, where CLARUS must keep the AES keys (whereas it keeps nothing in the other protocols). Temporary storage is of the same order of magnitude for Protocols 1.1, 1.2 and 2.1 and really small for CLARUS; instead, the benchmark Protocols 5 and 6 require large temporary storage on the CLARUS side.

Table 9 shows the computation and communication costs incurred by the execution of the above mentioned protocols. Communication takes a very substantial time both at the sender and at the receiver: obviously, the receiver cannot process the data until he receives them, and the sender needs to wait for the receiver's acknowledgment of receipt before carrying on. The communication between AWS and CLARUS is slow, in part because the Amazon services are geographically distant and in part because we used a free-of-charge instance. Note that, although CLARUS receives only scalars in Protocols 1.1, 1.2 and 2.1, its communication cost is greater than for Alice, who sends and receives vectors. The explanation is that: i) reading/sending between the CSPs (Alice and Bob) is faster than between the CSPs and CLARUS (because Alice and Bob are located in the same cloud system, AWS in one case or CERSEI in the other case); ii) in the CLARUS-CSP communication the time to send a scalar is dominated by the time to establish the channel. Protocols 1.1, 1.2 and 2.1 have all similar costs for CLARUS. In Protocol 2.1, the computational and communication costs for Alice and Bob are greater than in Protocols 1.1 and 1.2, because in the former protocol computations are performed over a field $\mathbb{F}_N$, where N is big and the representation of numbers takes 65 bytes instead of 8.

If we aggregate costs for all parties involved, Protocol 1.1 is the most efficient one, closely followed by Protocol 1.2 (note that these two protocols do not

Table 9: **(Execution costs for Protocols 1.1, 1.2, 2.1, 5 and 6)**: Private vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ have length $n = 20,640$. The representation of the numbers takes $\gamma = 8$ bytes. Elements in the field $\mathbb{F}_N$ used by the Paillier cryptosystem have 65 bytes. We indicate the absence of a cloud with "$-$". Times are given in seconds.

AWS	Computational cost (s.)				Communication cost (s.)			
	Alice	Bob	CLARUS	Charlie	Alice	Bob	CLARUS	Charlie
Prot. 1.1	8.3×10^{-3}	8.5×10^{-3}	3.3×10^{-5}	1.4×10^{-3}	0.05	0.05	0.40	0.06
Prot. 1.2	1.7×10^{-2}	1.6×10^{-2}	4.1×10^{-5}	1.4×10^{-3}	0.06	0.06	0.40	0.06
Prot. 2.1	34.1	121.6	9.1×10^{-5}	–	0.15	0.31	0.40	–
Prot. 5	0	0	1.5×10^{-3}	–	0.1	0.1	0.65	–
Prot. 6	0	–	0.8	–	2.2	–	2.62	–
CERSEI	Computational cost (s.)				Communication cost(s.)			
	Alice	Bob	CLARUS	Charlie	Alice	Bob	CLARUS	Charlie
Prot. 1.1	8.1×10^{-3}	7.8×10^{-3}	3.1×10^{-5}	1.4×10^{-3}	0.04	0.04	0.20	0.04
Prot. 1.2	5.7×10^{-3}	5.8×10^{-3}	3.7×10^{-5}	1.4×10^{-3}	0.04	0.04	0.20	0.04
Prot. 2.1	29.3	120.9	8.4×10^{-5}	–	0.07	0.08	0.20	–
Prot. 5	0	0	1.9×10^{-3}	–	0.02	0.01	0.23	–
Prot. 6	0	–	0.7	–	0.3	–	0.45	–

require any set-up). These results are consistent with the analytical comparison in Section 7.1.

8.2. Comparison for the sharing-resistant alternatives

In this section, Protocol 3 (sharing-resistant without background knowledge) and Protocol 4 (sharing-resistant with background knowledge) are compared with the benchmark Protocol 6.

Table 10 shows the storage, computation and communication costs for the aforementioned protocols. CLARUS needs significantly less storage, computation and communication resources in Protocols 3 and 4 than in the benchmark Protocol 6. In fact, once the set-up phase is completed, the sharing-resistant protocols are extremely convenient for CLARUS, because nearly all the computations are performed by the cloud and CLARUS only needs to do very little work. AWS and CERSEI give results that are consistent with the analytical comparison in Sections 7.2 and 7.3.

9. Conclusions and future work

We have presented several protocols (two of them existing, two of them variants, and four of them new, plus the two trivial benchmark protocols) for outsourcing to untrusted clouds two basic operations on sensitive data vectors: scalar products and matrix products. Based on those operations, more complex data analyses can be performed, such as correlations and contingency tables. The goal is to minimize the amount of work that needs to be performed locally by the controller, who wants to use the cloud as much as possible to compute on her outsourced sensitive data. For the sake of flexibility and efficiency, we have

Table 10: **(Costs for Protocols 3, 4 and 6 for AWS and CERSEI)**: $\boldsymbol{X}$ is a $20,640 \times 2$ matrix containing the original data set. The components in the vectors and matrices take $\gamma = 8$ bytes. The storage is given in bytes (B) and the computational and communication costs are given in seconds (s).

	Protocol 3			Protocol 6		Protocol 4	
	Alice (C_0)	Bob (C_j)	CLARUS	Alice	CLARUS	Alice	CLARUS
Long-term(B)	3.3×10^5	3.3×10^5	32	3.3×10^5	0	3.6×10^6	8
Temporary(B)	0	0	64	0	6.6×10^5	124	160
AWS	Protocol 3			Protocol 6		Protocol 4	
	Alice (C_0)	Bob (C_j)	CLARUS	Alice	CLARUS	Alice	CLARUS
$\mathbf{X^T X}$ comp.(s)	0.2	0	4.4×10^{-5}	0	0.9	0.05	2.03×10^{-5}
$\mathbf{X^T X}$ comm.(s)	1.2×10^{-4}	0	0.4	2.2	2.6	3.6×10^{-4}	0.4
CERSEI	Protocol 3			Protocol 6		Protocol 4	
	Alice (C_0)	Bob (C_j)	CLARUS	Alice	CLARUS	Alice	CLARUS
$\mathbf{X^T X}$ comp.(s)	0.08	0	4.2×10^{-5}	0	0.7	0.03	1.9×10^{-5}
$\mathbf{X^T X}$ comm.(s)	5.4×10^{-5}	0	0.2	0.3	0.5	8.2×10^{-5}	0.2

considered non-cryptographic methods for data protection, such as data splitting and anonymization, rather than the heavier fully homomorphic encryption (e.g. [17]). A distinguishing feature of our approach is that the outsourced data on which the clouds compute retain some of the utility of the original data, which entails added value with respect to outsourcing encrypted or otherwise gibberish data.

If clouds can be assumed not to share information (perhaps because they do not know each other), data splitting is probably the best choice, due to simplicity and flexibility. We have proposed four protocols to compute on split data. In case clouds can share information but have no side knowledge on the original data set, we have proposed a sharing-resistant protocol based on orthogonal noise matrices that shifts most of the computational burden to the clouds. Finally, for the worst case, in which clouds share information and have side knowledge allowing them to recognize the original data set, we have proposed a sharing-resistant protocol relying on noise matrices derived via anonymization. Although the latter protocol is heavier, it still substantially relieves the controller (CLARUS) in computational terms. We have provided complexity analyses and benchmarking for all proposed protocols, in order to show their computational advantages for the outsourcing controller. Further, we have provided experimental evidence that the new protocols take less effort from CLARUS than the benchmark protocols consisting of downloading and local processing.

In this way, clouds are not only used to store sensitive data, but also to perform computations on these data in a privacy-aware manner. This is especially interesting for large sensitive data sets.

Scalar products and matrix products are useful for correlations and contingency tables. While these are important analyses, many other analytical needs exist. Future research will be directed at broadening the range of data analyses that can be performed on protected sensitive data stored in the cloud.

Acknowledgments and disclaimer

Thank go to Aida Calviño for preliminary discussions on some ideas presented in the paper, and to Mónica del Carmen Muñoz for help with the experimental part. The following funding sources are gratefully acknowledged: European Commission (projects H2020 644024 "CLARUS" and H2020 700540 "CANVAS"), Government of Catalonia (ICREA Acadèmia Prize to J. Domingo-Ferrer and grant 2014 SGR 537), Spanish Government (projects TIN2014-57364-C2-R "SmartGlacis" and TIN2016-80250-R "Sec-MCloud"). The authors are with the UNESCO Chair in Data Privacy, but the views in this paper are their own and do not necessarily reflect those of UNESCO.

Appendix A. Finite field for the computations

Protocol 2 and its variants work in a finite field $\mathbb{F}_N$, i.e. given $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ two private n-vectors, we are computing $\boldsymbol{x}^T\boldsymbol{y} \mod (N)$. If we do not want the result to be modified by the modulus, it must hold that $N > \boldsymbol{x}^T\boldsymbol{y}$. Let $M_x = \max_{x_i \in x} x_i$ be the maximum value belonging to $\boldsymbol{x}$, $M_y = \max_{y_i \in y} y_i$ the maximum value belonging to $\boldsymbol{y}$ and $M = \max\{M_x, M_y\}$. It is sufficient to choose $N > nM_xM_y$. To chose such an N, we suggest that, before the protocol, Bob send $M'_y > M_y$ to Alice who chooses $M'_x > M_x$, and then picks $N > nM'_xM'_y$.

Alternatively, Alice can one-sidedly choose a very large N without Bob's input (a 1024-bit N is a common choice with Paillier's cryptosystem). In Protocol 2.2 the public key generation and hence this one-sided choice would be done by CLARUS.

The choice of a very large N allows decreasing the computational cost of the set-up phase of Protocols 2, 2.1 and 2.2 (reading the vectors is not necessary anymore and so the set-up phase has $O(1)$ cost). On the other hand, the computational cost of the scalar product in all three protocols is considerably increased, because a larger N means larger keys and ciphertexts, which make cryptographic operations slower.

Appendix B. Computation of the invertible matrix $\boldsymbol{P}$

The probability of obtaining an invertible matrix $\boldsymbol{P}$ (needed in the set-up phase of Protocol 3) if we randomly choose its elements depends on the cardinality of the field we are working in.

Lemma 1. *Let $\mathbb{K}$ be a field where the elements of a random $m \times m$ matrix $\boldsymbol{P}$ are chosen. If $|\mathbb{K}| = N$, then*

$$\Pr(\boldsymbol{P} \textit{ is invertible}) > 1 - \frac{1}{N-1} + \frac{1}{(N-1)N^m}. \tag{B.1}$$

Proof. $\boldsymbol{P}$ has m column vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$. In order for $\boldsymbol{P}$ to be invertible, they must be linearly independent. We have:

- $\Pr(\boldsymbol{v}_1$ is the null vector$) = \frac{1}{N^m}$;
- $\Pr(\boldsymbol{v}_2$ is linear combination of $\boldsymbol{v}_1) \leq \frac{N}{N^m}$;
- $\Pr(\boldsymbol{v}_3$ is linear combination of $\boldsymbol{v}_1, \boldsymbol{v}_2) \leq \frac{N^2}{N^m}$;
- ...
- $\Pr(\boldsymbol{v}_t$ is linear combination of $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1}) \leq \frac{N^{m-1}}{N^m}$.

Hence, $\Pr(\boldsymbol{P}$ is not invertible$) \leq \sum_{i=0}^{m-1} \frac{N^i}{N^m} = \frac{N^m-1}{(N-1)N^m}$ and then

$$\Pr(\boldsymbol{P} \text{ is invertible}) > 1 - \frac{N^m - 1}{(N-1)N^m} = 1 - \frac{1}{N-1} + \frac{1}{(N-1)N^m}.$$

□

We suggest to obtain $\boldsymbol{P}$ by randomly picking an $m \times m$ matrix over $\mathbb{K}$ and trying to invert it. This will work with probability lower-bounded by Expression (B.1). As N grows, the lower bound approaches 1, so if we take a sufficiently large $\mathbb{K}$, a single attempt is very likely to suffice to find $\boldsymbol{P}$. If the first attempt fails, one can always try again.

Appendix C. Orthogonal complement of a matrix

In Protocol 3 (set-up phase), each cloud C_i needs to find an $n \times m$ matrix $\boldsymbol{E}_i$ orthogonal to $\boldsymbol{X}'$. Following [22], we suggest to find $Ort(\boldsymbol{X}')$ by using the QR-decomposition of $\boldsymbol{X}'$, where $\boldsymbol{Q}$ is an $(n \times n)$ orthonormal matrix and $\boldsymbol{R}$ is an $n \times m$ upper-triangular matrix. If we split $\boldsymbol{Q} = [\boldsymbol{Q}_1, \boldsymbol{Q}_2]$, where $\boldsymbol{Q}_1$ is $(n \times m)$ and $\boldsymbol{Q}_2$ is composed of $n - m$ orthogonal vectors to $\boldsymbol{X}'$, it is possible to select m columns out of $\boldsymbol{Q}_2$ to generate $\boldsymbol{E}_i$. Note that $n \gg m$, so m columns can be selected out of $n - m$ and each cloud is likely to select a different set of m columns.

If $n - m > m$, but it does not hold that $n \gg m$, then m random linear combinations of the columns of $\boldsymbol{Q}_2$ are preferable to using the columns themselves: since Protocol 3 (set-up phase) will be used by all the clouds $C_1, C_2, \ldots, C_t$, we want to avoid the possibility that different clouds obtain the same $\boldsymbol{E}_i$.

References

[1] Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: a distributed architecture for secure database services. In CIDR 2005, pp. 186–199 (2005).

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Communications of the ACM, 53(4):50–58 (2010).

[3] Atallah, M. J., Frikken, K. B.: Securely outsourcing linear algebra computations. In 5th ACM Symposium on Information, Computer and Communications Security – ASIACCS 2010, pp. 48-59. ACM (2010).

[4] California housing data set (cadata.txt), Louisiana State University, 2006. http://statweb.lsu.edu/faculty/li/data/

[5] Calviño, A., Ricci, S., Domingo-Ferrer, J.: Privacy-preserving distributed statistical computation to a semi-honest multi-cloud. In IEEE Conf. on Communications and Network Security – CNS 2015, pp. 506-514. IEEE (2015).

[6] Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Transactions on parallel and distributed systems, 25(1):222-233 (2014).

[7] Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and encryption to enforce privacy in data storage. In Computer Security – ESORICS 2007, Lecture Notes in Computer Science, vol. 4734, pp. 171–186, Springer (2007).

[8] Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Selective data outsourcing for enforcing privacy. Journal of Computer Security, 19(3):531–566 (2011).

[9] CLARUS - A Framework for User Centred Privacy and Security in the Cloud, H2020 project (2015-2017). http://www.clarussecure.eu

[10] Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., Zhu, M.: Tools for privacy preserving distributed data mining. ACM SIGKDD Explorations Newsletter, 4(2):28–34 (2002).

[11] Domingo-Ferrer, J., Muralidhar, K.: New directions in anonymization: permutation paradigm, verifiability by subjects and intruders, transparency to users. Information Sciences, 337-338:11–24 (2016).

[12] Du, W., Han, Y., Chen, S.: Privacy-preserving multivariate statistical analysis: linear regression and classification. In Proc. of the 2004 SIAM International Conference on Data Mining, pp. 222–233 (2004).

[13] Du, W., Zhan, Z.: A practical approach to solve secure multi-party computation problems. In Proceedings of the 2002 Workshop on New Security Paradigms, pp. 127–135. ACM (2002).

[14] Dubovitskaya, A., Urovi, V., Vasirani, M., Aberer, K., Schumacher, M.: A cloud-based e-health architecture for privacy preserving data integration. In ICT Systems Security and Privacy Protection. Springer, pp. 585–598 (2015).

[15] Durstenfeld, R: Algorithm 235: random permutation. Communications of the ACM 7(7):420 (1964).

[16] Ganapathy, V., Thomas, D., Feder, T., Garcia-Molina, H., Motwani, R.: Distributing data for secure database services. Transactions on Data Privacy, 5(1):253–272 (2012).

[17] Gentry, C. Fully homomorphic encryption using ideal lattices. In 41st ACM Symposium on Theory of Computing – STOC 2009, pp. 169178. ACM, 2009.

[18] Goethals, B., Laur, S., Lipmaa, H., Mielikäinen, T.: On private scalar product computation for privacy-preserving data mining. In Information Security and Cryptology – ICISC 2004, Lecture Notes in Computer Science, vol. 3506, pp. 104–120, Springer (2005).

[19] Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Nordholt, E., Spicer, K., de Wolf, P.: Statistical Disclosure Control. John Wiley & Sons (2012).

[20] Ioannidis, I., Grama, A., Atallah, M.: A secure protocol for computing dot-products in clustered and distributed environments. In Proceedings of the International Conference on Parallel Processing. IEEE, pp. 379–384 (2002).

[21] Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In 2012 ACM Conference on Computer and Communications Security – ACM CCS 2012, pp. 965–976. ACM 2012.

[22] Karr, A., Lin, X., Sanil, A., Reiter, J.: Privacy-preserving analysis of vertically partitioned data using secure matrix products. Journal of Official Statistics, 25(1):125-138 (2009).

[23] Lei, X., Liao, X., Huang, T., Heriniaina, F.: Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud. Information Sciences 280:205-217 (2014)

[24] Lei, X., Liao, X., Huang, T., Li, H., Hu, C.: Outsourcing large matrix inversion computation to a public cloud. IEEE Transactions on Cloud Computing 1(1):78-87 (2013)

[25] Liu, F., Ng, W.K., Zhang, W. Encrypted scalar product protocol for outsource data mining. In IEEE CLOUD 2014, pp. 336-343. IEEE (2014).

[26] Nassar, M., Erradi, A., Sabry, F., Malluhi, Q. M.: Secure outsourcing of matrix operations as a service. In IEEE CLOUD 2013, pp. 918-925. IEEE (2014).

[27] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In Advances in Cryptology – EUROCRYPT '99, Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999).

[28] Ren, K., Wang, C., Wang, Q.: Security challenges for the public cloud. IEEE Internet Computing, 16(1):69–73 (2012).

[29] Sun, Y., Yu, Y., Li, X., Zhang, K., Qian, H., Zhou, Y.: Batch verifiable computation with public verifiability for outsourcing polynomials and matrix computations. In Australasian Conference on Information Security and Privacy – ACISP 2016, Lecture Notes in Computer Science, vol. 9722, pp. 293–309. Springer (2016).

[30] Trouessin, G.: Traitements fiables de données confidentielles par fragmentation-redondance-dissémination. Ph.D. dissertation, Université de Toulouse 3 (1991).

[31] Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In Proc. of the 9th International Conference on Knowledge Discovery and Data Mining – KDD 2003, pp. 206-215. ACM (2003).

[32] Wang, I.-C., Shen, C.-H., Hsu, T.-S., Liao, C.-C., Wang, D. W., Zhan, J.: Towards empirical aspects of secure scalar product. IEEE Trans. Systems, Man and Cybernetics, Part C, 39(4):440–447 (2009).

[33] Weiss, G.: Data mining in the real world: experiences, challenges, and recommendations. In DMIN, pp. 124–130 (2009).

[34] Yang, Q., Wu, X.: 10 challenging problems in data mining research. International Journal of Information Technology & Decision Making, 5(4):597–604 (2006).