Chiaroscuro: Transparency and Privacy for Massive Personal Time-Series Clustering

Tristan Allard, Georges Hébrail, Florent Masseglia, Esther Pacitti

Chiaroscuro: Transparency and Privacy for Massive Personal Time-Series Clustering

Tristan Allard[†] Georges Hébrail[*] Florent Masseglia[#] Esther Pacitti[+]
[†] IRISA & Univ. Rennes 1, France
[*] EDF R&D, Clamart, France
[#,+] Inria & Lirmm, Univ. Montpellier, France
first.last@{[†]irisa.fr, [*]edf.fr, [#]inria.fr, [+]lirmm.fr}

ABSTRACT

The advent of on-body/at-home sensors connected to personal devices leads to the generation of fine grain highly sensitive personal data at an unprecendent rate. However, despite the promises of large scale analytics there are obvious privacy concerns that prevent individuals to share their personnal data. In this paper, we propose *Chiaroscuro*, a complete solution for clustering personal data with strong privacy guarantees. The execution sequence produced by Chiaroscuro is massively distributed on personal devices, coping with arbitrary connections and disconnections. Chiaroscuro builds on our novel data structure, called *Diptych*, which allows the participating devices to collaborate privately by combining encryption with differential privacy. Our solution yields a high clustering quality while minimizing the impact of the differentially private perturbation. Chiaroscuro is both correct and secure. Finally, we provide an experimental validation of our approach on both real and synthetic sets of time-series.

Categories and Subject Descriptors: H.2.8 Database Management: Data Mining; K.4.1 Computers and Society: Privacy; H.2.4 Database Management: Distributed Databases;

Keywords: differential privacy; secure multi-party computation; clustering; k-means; time-series; gossip; sensors.

1. INTRODUCTION

Nowadays, individuals are able to monitor various biometric indicators (through, *e.g.,* heart-rate or blood-flow sensors, accelerometers, connected scales) for health, sports, or well-being reasons, as well as various consumption indicators (through, *e.g.,* smart-meters or smart-plugs for electricity or water consumption). This results in the production of *big personal data* in the form of sequences of time-stamped real values, called *time-series*, stored in the personal devices of the individuals.

Such wealth of personal time-series opens up exciting opportunities to learn valuable knowledge. Cluster analysis, also called *clustering*, aims at forming groups of data (or *clusters*) such that similar data appear in the same group and dissimilar data appear in different groups. Clustering is widely used in various application domains, *e.g.*, medicine, genetics, marketing, or social sciences. Clustering could, for example, allow a household to discover that its electrical price plan is not suited to its usage, compared to the price plan usually chosen by those having similar electrical consumption habits.

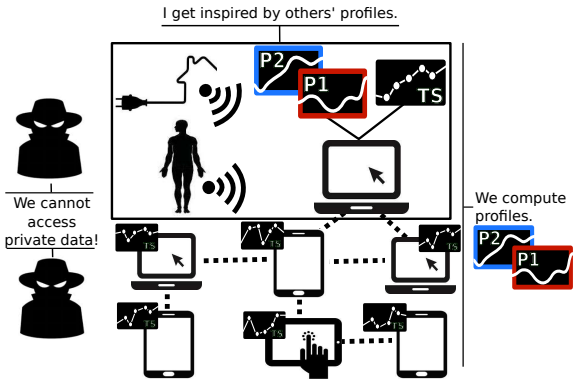


Figure 1: Collaborative Clustering of Massive Personal Time-Series with Privacy Guarantees

However, the quantity of information about individuals that fine grain personal time-series convey becomes worrying. Indeed, a personal time-series may disclose a wide and arbitrary variety of personal information, potentially leading to the identification and exposure of sensitive data. For example, a time-series containing the electrical consumption of a household recorded at the typical sampling rate of one measure every minute is a precise dissection of the in-home activities. At such fine grain, domestic appliances exhibit almost-unique electrical signatures that can then be identified accurately within the time-series [31, 32]. Demographic information, such as number or ages of individuals, may be inferred based on the volumes of electricity consumed and the times of consumption; For instance, a health status may be revealed by the presence of a clinic bed; even a religious orientation may become highly probable when observing regular electrical patterns (*e.g.*, the Jewish Shabbat involves a reduced electrical activity between Friday and Saturday sunsets). Similar conclusions are drawn when time-series come from different application domains (*e.g.*, physiological measures).

The usual approach for clustering time-series is centralized; it consists in copying the time-series from personal devices to a central server, assumed to be trustworthy, which is then in charge of performing clustering. However, although entrusting a single entity with this wealth of data is practical, it is hazardous in

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2749453 .

an untrusted world. Millions of personal records are getting exposed each year due to outsider attacks, insider attacks, or basic negligence (*e.g.*, stolen laptop, lost post mail). Such cases feed the news on a daily basis (see, *e.g.*, `http://datalossdb.org`). According to recent reports (*e.g.*, [3, 34]), this trend is growing rapidly, backed up by the ever-increasing amount of personal data that is centralized. Thus, individuals are more and more concerned by the systematic collection and centralization of their personal data[1]. As a result, the centralized approach not only introduces an additional risk to data privacy but also an additional obstacle to large-scale personal data analysis.

In this paper, we address the problem of clustering personal time-series that are massively distributed on personal devices without jeopardizing their privacy (see Figure 1). This problem is not addressed satisfactorily by related work. First, the common use of random parts or of generic secure multi-party computation techniques [5, 17, 18, 19, 20] severely questions their ability to scale with the number of participating devices and cope with arbitrary connections/disconnections. Second, the straightforward disclosure of data-dependent information proposed in some work, *e.g.*, [22, 35], jeopardizes their privacy guarantees. When addressing this problem, we set the following requirements for the clustering solution:

- **R1 Transparent execution:** clustering must be performed by the set of participating devices in a collaborative manner;
- **R2 Private execution:** no information threatening the privacy of individuals must leak from the collaborative execution of clustering;
- **R3 Quality:** the quality of clustering must be comparable to that of a centralized clustering;
- **R4 Practicality:** the performance of clustering must be acceptable in contexts where the number of participating devices is in the order of several millions;

We propose *Chiaroscuro*[2], a complete solution for cluster analysis over personal time-series with strong privacy guarantees. Chiaroscuro comes with a privacy-preserving clustering algorithm fully decentralized on collaborating personal devices (also called *participants* in the following) that fulfills altogether the above requirements. Through requirement R2, Chiaroscuro relaxes the traditional *desideratum* that usually rules the security of distributed algorithms, allowing *differentially private* intermediate results to be revealed during the execution. The use of complex secure multi-party computation protocols is thus avoided by operating on cleartext data, while their privacy remains guaranteed by satisfying differential privacy.

We make the following contributions:

1. A massive distribution of the execution sequence on the set of participating personal devices based on gossip aggregation algorithms, thus fulfilling R1;
2. A data structure, called *Diptych*, which allows honest-but-curious participants to collaborate while never exporting any personal time-series out of any host device in a form that is neither homomorphically encrypted nor differentially private, thus fulfilling R2;
3. A set of heuristics designed to minimize the impact of differential privacy on the clustering quality, thus fulfilling R3;
4. A thorough experimental evaluation on both real and synthetic sets of time-series, which shows that R3 and R4 are fulfilled;

The paper is organized as follows. Section 2 defines more precisely the problem we tackle. Section 3 introduces necessary concepts and techniques. Section 4 describes Chiaroscuro with its data structure and the execution sequence, and shows its correctness and security. Section 5 deals with the quality of the clustering obtained. Section 6 experimentally validates the quality and practicality of Chiaroscuro. Section 7 discusses related work and Section 8 concludes.

2. PROBLEM DEFINITION

2.1 Time-Series and Clustering

Chiaroscuro aims at clustering time-series. A time-series is a sequence of real-valued variables. A usual representation of a time-series s having length n is given as an ordered set of real numbers: $s =< s[1]\ s[2]\ ...\ s[n] >$. Let S be a set of t times series, where each time-series s in S has length n, then S can be represented as a matrix, as follows:

$$S = \begin{bmatrix} s_1 =< s_1[1] & \cdots & s_1[n] > \\ \vdots & \vdots & \vdots \\ s_t =< s_t[1] & \cdots & s_t[n] > \end{bmatrix} \quad (1)$$

Clustering is an unsupervised learning task that aims at building a disjoint set of clusters on a dataset. Let k be a parameter given by the end-user ($1 < k < t$) standing for the number of clusters to build. A disjoint set of clusters $\zeta = \{\zeta[1], \cdots, \zeta[k]\}$ is obtained by a horizontal partitioning of S (*i.e.,* $\cup_{i=1}^{k} \zeta[i] = S$ and $\forall i,j/i \neq j, \zeta[i] \cap \zeta[j] = \emptyset$). The goal of time-series clustering is to build such a partitioning, where similar series are grouped together, while dissimilar series are separated in different clusters. The result of clustering might be described by extension (enumerating the content of each cluster) or by intension with descriptors (*e.g.,* for each cluster, statistics, or a centroid such as a mean value).

2.2 Participants

Individuals participate in Chiaroscuro through their personal devices (*e.g.,* tablets, smartphones, laptops). In the rest of this paper, we call *participant* a computing node that participates in Chiaroscuro's execution sequence. We abstract participants by a set of assumptions about the resources they provide and the attack model they follow.

Resources. We expect a personal device to provide average resources: a multi-core CPU clocked at several GHz, a GB-sized RAM, and a GB-sized non-volatile memory (at least). It is also able to connect to the Internet at a high rate (*e.g.,* 1-10 Mb/s). However, since it is controlled by its owner, it does not provide any guarantee of availability: its connections and disconnections are arbitrary.

Attack model. A participant is called *honest-but-curious* when it follows strictly the execution sequence but exploits in any computationally-feasible way the information made available during the execution for inferring personal data. Moreover participants may collude together up to a reasonable bound initially fixed denoted by τ in the following.

2.3 Correctness and Security

Correctness. Chiaroscuro is *correct* iff (1) it terminates, and (2) outputs at least one centroid.

Secrets. The privacy guarantees of modern encryption/perturbation schemes rely on the secrecy of a small set of infor-

[1]Flagship examples include the Google Health, Google PowerMeter, and Microsoft Hohm products, abandoned because of insufficient adoption.

[2]Chiaroscuro is named after the conjunction of clarity - due to R1 - with obscurity - due to R2.

k-means Algorithm	
k	Initial number of centroids.
$\mathcal{C}_{init}$	Initial set of centroids.
θ	Convergence threshold.
Epidemic Aggregation Algorithm	
Λ	Initial local view.
n_e	Number of exchanges.
Cryptographic/Privacy Schemes	
χ	Public encryption key.
κ_i	Private i^{th} key-share.
τ	Key-shares threshold.
ϵ	Differential privacy level.
δ	Minimum probability to guarantee differential privacy.
n_ν	Number of noise-shares to be summed up together.

Table 1: Building Blocks' Initialization Parameters

mation such as the decryption key. We call them *secrets* and denote by Ξ the set of secrets. We present them in details in the next section.

Security.[3] Chiaroscuro is *secure* against honest-but-curious participants iff (1) no participant learns anything about the input time-series that has not been perturbed by a differentially-private mechanism (2) the uncertainty of a colluding participant over the set of secrets decreases gracefully with the number of collusions (which is bounded by the threshold τ).

3. PRELIMINARIES

In this section, we introduce basic concepts from clustering, distribution, and privacy-preserving techniques that are necessary in this work. Table 1 gives the initial parameters[4].

3.1 k-means Algorithm

The goal of k-means [28] is to propose k clusters that minimize an objective function. Usually, the objective function is the *intra-cluster inertia* (Definition 1), which measures the homogeneity of the set of time-series within clusters based on their average values, also called *centroids*. Let $\mathcal{C}$ denote the set of centroids. The centroid $\mathcal{C}[i] \in \mathcal{C}$ of the cluster $\zeta[i] \in \zeta$ is computed as follows: $\mathcal{C}[i] = <\mathcal{C}[i,1] \cdots \mathcal{C}[i,n]>$ where $\mathcal{C}[i,j] = \frac{1}{|\zeta[i]|} \sum_{\forall s \in \zeta[i]} s[j]$. Each centroid stands for the description by intension of its corresponding cluster.

DEFINITION 1 (INERTIA). *The* intra-cluster inertia *of a cluster $\zeta[i]$ is the weighted sum of the squared euclidean distances between the time-series in $\zeta[i]$ and the centroid $\mathcal{C}[i]$. The intra-cluster inertia of a set of clusters ζ, denoted by q^{ζ}_{intra}, is the sum of the intra-cluster inertia of $\zeta_i \in \zeta$: $q^{\zeta}_{intra} = \frac{1}{t} \cdot \sum_{i=1}^{k} \sum_{\forall s \in \zeta[i]} ||\mathcal{C}[i] - s||^2$. The* inter-cluster inertia *of a set of clusters ζ measures their heterogeneity. It is the weighted sum of the squared euclidean distances between the clusters' centroids and the centroid of the complete set of time-series. We denote it by q^{ζ}_{inter} and compute it as : $q^{\zeta}_{inter} = \sum_{i=1}^{k} \frac{|\zeta[i]|}{t} \cdot ||\mathcal{C}[i] - g||^2$, where g is the centroid of the complete set of time-series. The* full inertia *of the dataset is the addition of the intra-cluster inertia to the inter-cluster inertia : $q^{\zeta} = q^{\zeta}_{intra} + q^{\zeta}_{inter}$. It is a constant value.*

[3]The formal statement of our security model is in the Appendix.

[4]Participating devices can get the set of initial parameters, *e.g.,* as usual by downloading it at initialization time from a bootstrap server (which does not participate any further in the execution sequence), possibly after an authentication step.

Given k ($1 < k < t$, with t the number of time-series), k-means initially sets the k initial centroids $\mathcal{C}_{init}$ to arbitrary values and the k clusters $\zeta[i] \in \zeta$ to empty sets. Then, it proceeds as follows:

1. **Assignment step:** For each $s \in S$, get its closest centroid $\mathcal{C}[i] \in \mathcal{C}$ and assign it to the corresponding cluster $\zeta[i] \in \zeta$.
2. **Computation step:** For each cluster $\zeta[i] \in \zeta$, compute $\mathcal{M}[i] \in \mathcal{M}$, its corresponding candidate centroid (also called its *mean* below).
3. **Convergence step:** If the distance between the centroids $\mathcal{C}$ and the means $\mathcal{M}$ is greater than θ, then set $\mathcal{C} \leftarrow \mathcal{M}$ and loop to the assignment step (Step 1), otherwise return $\mathcal{M}$.

3.2 Epidemic Aggregation Algorithms

Gossip protocols are lightweight fully decentralized protocols executed within large sets of participants. They simply consist in periodical point-to-point exchanges of information between participants. An initial list of IP addresses of some random participants, called the *local view*, and denoted by Λ in the following, bootstraps the exchanges. This simplicity makes gossip-based protocols easy to implement and highly scalable. Moreover, high levels of robustness and resistance to failures can be achieved by merging the local views at each exchange between participants [25]. *Epidemic aggregation algorithms* follow the gossip principles for computing approximations of global aggregates, such as sums or averages, over massively distributed data [23, 21]. Each participant holds its own approximation of the global aggregate, called *the local state*, and updates it, at each exchange, with the local state of the other participant through *the local update rule*. The approximation error depends on the number of gossip exchanges per participant and is guaranteed to converge to zero exponentially fast [21, 23]. Moreover, since the number of exchanges per participant is a parameter (denoted by n_e), it can be set so that the approximation error is as small as desired (*i.e.*, typically several orders of magnitude lower than the exact aggregate value [21], as illustrated in our experiments and shown in Appendix B).

Chiaroscuro is based on the epidemic computation of sums. In a context where the data is not sensitive, the epidemic sum algorithm proceeds as follows [23]. The local state of a participant i is made of a *sum*, denoted by σ_i, and a *weight*, denoted by ω_i. Each participant initializes σ_i to its local data and ω_i to zero[5] and starts its exchanges, updating σ_i and ω_i at each exchange. The local update rule simply consists in adding half of the local σ_i value (resp. ω_i) to half of the other participant's σ_j value (resp. ω_j). The local estimate of the global sum is σ_i / ω_i.

3.3 Encryption and Perturbation Schemes

3.3.1 Additively-Homomorphic Encryption Scheme

Chiaroscuro is independent of any specific encryption scheme provided that it satisfies the following properties:

1. **Semantic security[14]**: Stated informally, semantic security implies that, given a ciphertext, the public encryption key, and, some information about the plaintext, no probabilistic polynomial-time algorithm is able to gain additional knowledge on the plaintext. In particular, semantically secure schemes are not deterministic ;
2. **Additively-homomorphic:** Two encrypted integers can be added together to yield a correct encryption of their addition. Let a and b be two integers, χ be the encryption key and κ the decryption key, and $+_h$ be the homomorphic addition operator, then: $\mathtt{D}_\kappa(\mathtt{E}_\chi(a) +_h \mathtt{E}_\chi(b)) == a + b$;

[5]One participant, no matter which one, must initialize ω_i to one. This is easily done in practice, *e.g.,* through a dedicated machine acting as a participant.

3. **Non-interactive threshold decryption:** The decryption key is a set of n_κ *key-shares* $\{\kappa_i\}$ so that decrypting a message requires to decrypt it partially by at least $\tau \leq n_\kappa$ distinct key-shares. Each partial decryption can be performed independently of the others;

The Damgard-Jurik encryption scheme [8] is an instance of such schemes:

1. **Encryption key:** The public encryption key is $\chi=(n,g)$, where n is an RSA modulus (the product of two large prime numbers) and g an element of the computation space, *i.e.,* the finite field $\mathbb{Z}^*_{n^{s+1}}$ (with $s>0$ a positive integer);
2. **Decryption key-shares:** Each decryption key-share κ_i is a point of a polynom of degree τ (so that τ points are necessary for knowing the polynom), where $\tau \leq n_\kappa$ is the key-shares threshold. The polynom is part of the set of secrets Ξ;
3. **Encryption:** The encryption of an integer a is $\mathtt{E}_\chi(a) = g^a \cdot r^{n^s} \mod n^{s+1}$ where $r \in \mathbb{Z}^*_{n^{s+1}}$ is a random number;
4. **Homomorphic addition:** The additively-homomorphic addition operator is simply the product between the two encrypted values: $\mathtt{E}_\chi(a) +_h \mathtt{E}_\chi(b) = \mathtt{E}_\chi(a) \times \mathtt{E}_\chi(b)$;
5. **Decryption:** Decryption is essentially performed by decrypting partially the encrypted value by at least $\tau \leq n_\kappa$ distinct key-shares, and by computing the product of the τ partial decryptions. A partial decryption consists basically in raising the encrypted value to a power that is a multiple of κ_i;

3.3.2 Differential Privacy and Laplace Perturbation

The ϵ-differential privacy model [10] (see Definition 2) requires that whatever the output of a (differentially-private) aggregation function, the probability that any given individual's data d was in the input is indistinguishable from the probability that d was not in the input. In our context, the aggregation function is the (time-series) sum, denoted by `Sum` in the following. It inputs a set of time-series, each of length n, and outputs their dimension-wise sum, *i.e.,* a single time-series of length n too where the variable at the i^{th} dimension is the sum of the variables at the i^{th} dimension of all the time-series input.

DEFINITION 2 (ϵ-DIFFERENTIAL PRIVACY FROM [10]). `Sum` *satisfies ϵ-differential privacy if, given the privacy parameter $\epsilon > 0$: $\mathtt{Pr}(\mathtt{Sum}(S_1) \in \mathcal{I}) \leq e^\epsilon \cdot \mathtt{Pr}(\mathtt{Sum}(S_2) \in \mathcal{I})$ for any set $\mathcal{I}$ and any set of time-series S_1 and S_2 such that S_2 can be obtained from S_1 by inserting or deleting one individual's time-series.*

In Chiaroscuro, the massive distribution of the differential privacy mechanism is based on approximate epidemic aggregation algorithms (see Section 3.2). In order to cope with the inherent approximation error, we relax the ϵ-differential privacy to a probabilistic variant, namely the (ϵ,δ)-probabilistic differential privacy[6] (Definition 3). The (ϵ,δ)-probabilistic differential privacy introduces a tradeoff between privacy and latency: together with other parameters, the value of δ determines the number of exchanges of the epidemic aggregation algorithms (see Appendix B).

DEFINITION 3 (PROBABILISTIC DIFFERENTIAL PRIVACY). `Sum` *satisfies (ϵ, δ)-probabilistic differential privacy if, given the privacy parameters $\epsilon > 0$ and $\delta \in]0,1]$, `Sum` satisfies ϵ-differential privacy with a probability greater than or equal to δ.*

[6]A probabilistic relaxation of differential privacy, similar in spirit to our variant, was also proposed in [29].

A crucial property of the (ϵ, δ)-probabilistic differential privacy model is its *composability*: a set of n independent aggregates, each satisfying (ϵ, δ)-probabilistic differential privacy, satisfies $(\sum_{i=1}^{n} \epsilon_i,\ \delta^n)$-probabilistic differential privacy. Hence, the ϵ privacy level initially fixed can be seen as *a privacy budget* to be distributed across the various aggregates to perturb.

The ϵ-differential privacy model can be satisfied by perturbing the true results of the aggregation function through the *Laplace mechanism* so that the impact of any individual on the result is overwhelmed by the perturbation introduced [12]. The perturbation, called *noise* in the rest of the paper, is chosen at random in a Laplace distribution centered at 0 with a scale factor that depends on the privacy parameter ϵ and on the *sensitivity* of the sum function, *i.e.,* the maximum impact that an insertion/deletion of an individual's time-series can incur on its output. The exact value of the noise is part of the set of secrets Ξ.

DEFINITION 4 (LAPLACE MECHANISM FROM [12]). *Let $\mathcal{L}(\lambda)$ denote a random variable which has a Laplace distribution with probability density function $f(x,\lambda) = \frac{1}{2\lambda} \cdot e^{-|x|/\lambda}$. Let S_1 and S_2 be any two sets of time-series such that S_2 can be obtained from S_1 by inserting or deleting one individual's time-series, and $[d_{min},d_{max}]$ be the possible range of each variable in a time-series. The Laplace mechanism consists in adding $\mathcal{L}(\max \|\mathtt{Sum}(S_1) - \mathtt{Sum}(S_2)\|_1/\epsilon) = \mathcal{L}(n * \max(|d_{min}|,|d_{max}|)/\epsilon)$ to each variable of the time-series output by* `Sum`.

Lemma 1 states that a Laplace random variable can be generated by summing together an arbitrary number of independently identically distributed random variables formally defined in Definition 5 and called *noise-shares*[7].

LEMMA 1 (DIVISIBILITY OF LAPLACE FROM [24]). *Let $\mathcal{L}(\lambda)$ denote a random variable that has a Laplace distribution with probability density function $f(x,\lambda) = \frac{1}{2\lambda} \cdot e^{-|x|/\lambda}$. Then the distribution of $\mathcal{L}(\lambda)$ is infinitely divisible. Furthermore, for every integer $n_\nu \geq 1$, $\mathcal{L}(\lambda) = \sum_{i=1}^{n_\nu} (\nu_i)$ where each ν_i is a noise-share independently generated as defined in Definition 5 .*

DEFINITION 5 (NOISE-SHARE). *A noise share ν_i is a random variable computed as follows: $\nu_i = \mathcal{G}_1(n_\nu,\lambda) - \mathcal{G}_2(n_\nu,\lambda))$, where n_ν is the total number of noise-shares to be summed up together, and $\mathcal{G}_1(n_\nu,\lambda)$ and $\mathcal{G}_2(n_\nu,\lambda)$ are i.i.d. random variables having gamma distribution with PDF $g(x,n_\nu,\lambda) = \frac{(1/\lambda)^{1/n_\nu}}{\gamma(1/n_\nu)} \cdot x^{\frac{1}{n_\nu}-1} \cdot e^{-x/\lambda}$ where $x \geq 0$.*

4. CHIAROSCURO

In designing Chiaroscuro, we must cope with highly constrained challenges related to clustering, distribution and privacy. In this section, we propose our solutions for each of these challenges and show that the complete resulting execution sequence is both correct and secure. For simplifying the presentation, we defer to Section 5 the issues related to quality, including the management of the privacy budget.

4.1 Diptych Data Structure

Parallelizability is a crucial property in our massively distributed context, making a clustering algorithm such as *k-means* [28] especially relevant. However, exploiting it without any security safeguard would jeopardize privacy irremediably. A strong rethinking of its execution is thus essential. A traditional

[7]This well-known mathematical property is also used in [1]. However, the fault-tolerance needed in our context and the absence of central entity yields a significantly different noise generation algorithm (see Sections 4 and 7).

centralized k-means iteration consists in (1) partitioning the dataset according to the k centroids input by assigning each time-series to the closest centroid, and (2) computing the centers of mass of the resulting partitions, *i.e.,* the k means. The k-means algorithm is thus essentially based on a twofold data structure made of the centroids on one side and the means on the other side.

Our approach fundamentally stems from a major recast of this data structure and, consequently, of the execution sequence that manipulates it. We design Chiaroscuro so that participants perform (1) the assignment step on *differentially private cleartext centroids* and (2) the computation step on *additively-homomorphic encrypted means*. The resulting data structure, consisting in the differentially private centroids and the encrypted means, is called *Diptych*. The means part of the Diptych structure must be designed carefully because it must support the epidemic sum algorithm, which relies on a specific weighted data representation (see Section 3), and cope with the fact that data is additively-homomorphic encrypted, which excludes performing any division before decryption. The Diptych data structure is detailed in Definition 6.

DEFINITION 6 (DIPTYCH DATA STRUCTURE). *A* Diptych *follows the form* $(\mathcal{C},\mathfrak{M})$ *where any* $\mathcal{C}[i] \in \mathcal{C}$ *is a cleartext perturbed centroid, and any* $\mathfrak{M}[i] \in \mathfrak{M}$ *is an encrypted mean.* $\mathfrak{M}[i]$ *is represented by* $(\mathfrak{f}=\mathtt{E}_\chi(\sigma_{\text{sum}}),\mathfrak{c}=\mathtt{E}_\chi(\sigma_{\text{count}}),\omega)$*, where* $\mathtt{E}$ *is the additively-homomorphic encryption scheme,* χ *the encryption key,* $\mathtt{E}_\chi(\sigma_{\text{sum}})$ *and* $\mathtt{E}_\chi(\sigma_{\text{count}})$ *denote the encrypted epidemic representations of the mean's sum (a time-series) and its count (a real value), respectively, with* ω *their complementary weight.*

We emphasize that any information that depends on the participant's data is either encrypted (the sum and count parts of the epidemic representation) or perturbed by a differentially private mechanism (the centroids). We let the weight appear harmlessly in the clear because it is independent of the data.

4.2 Execution Sequence

The Diptych data structure is key for Chiaroscuro's execution sequence. This sequence is iterative, fully distributed on each participant, and proceeds without any global synchronization (the late participants simply synchronize on the latest iteration during their gossip exchanges). Basically, each iteration computes the set of means in an encrypted manner, perturbs them by a differentially private mechanism without decrypting them, and finally decrypts them. It consists of the following steps performed in loop until the centroids converge (see Algorithm 1 for the full execution sequence):

1. **Assignment step:** the local participant assigns its data to the closest centroid in $\mathcal{C}$ and initializes (1) the corresponding encrypted mean in $\mathfrak{M}$ with its local time-series (dimension-wise) encrypted and (2) the other means with $k-1$ zero-valued time-series (dimension-wise) encrypted;
2. **Computation step:** Compute $\mathcal{M}$, the set of *perturbed* means of the new clusters:
   (a) Epidemic computation of the encrypted means;
   (b) Epidemic perturbation of the encrypted means;
   (c) Epidemic decryption of the perturbed encrypted means;
3. **Convergence step:** if the distance between $\mathcal{C}$ and $\mathcal{M}$ is greater than θ, then $\mathcal{C} \leftarrow \mathcal{M}$ and loop to the assignment step (Step 1), otherwise return $\mathcal{M}$.

The assignment and convergence steps are both performed on cleartext data, so they do not present strong challenges. This is the computation step, detailed in Algorithm 3, that concentrates the three unsolved points of the execution sequence: the computation of the encrypted means (Section 4.2.1), their differentially private perturbation (Section 4.2.2), and finally their decryption (Section 4.2.3)[8].

Algorithm 1: Full execution sequence run by each participant

Req.: **The local data:** a time-series s.
The security parameters: the public encryption key χ, a private decryption key-share κ_i, the differential privacy level ϵ, the number of noise-shares n_ν.
The epidemic parameters: an initial local view Λ, the required number of exchanges n_e.
The k-means parameters: the initial vector of k centroids $\mathcal{C}_{init}$ and the maximum number of iterations n_{it}^{max}.

1 Let $\mathcal{C}$ denote the vector of centroids, $\mathcal{M}$ denote the vector of intermediate perturbed decrypted means, $\mathfrak{M}.s$ the vector of the encrypted sums of means, and $\mathfrak{M}.c$ the vector of the encrypted counts of means: $\mathcal{C}=\mathcal{C}_{init}$, $\mathcal{M}=\{\varnothing\}$.
2 Let $cv=$`false` and $n_{it}=0$.
3 **while** $cv==$ *`false` and* $n_{it} \leq n_{it}^{max}$ **do**
4 **begin** *Assignment step:*
5 Compute the centroid that is the closest to s, $\mathcal{C}[j]$.
6 Initialize the encrypted means: $\mathfrak{M}[j].\mathfrak{f}=\{\mathtt{E}_\chi(s_i)\}_{\forall s_i \in s}$, $\mathfrak{M}[j].\mathfrak{c}=\mathtt{E}_\chi(1)$; and $\forall i \neq j : \mathfrak{M}[i].s=\{\mathtt{E}_\chi(0)\}$, $\mathfrak{M}[l].c=\mathtt{E}_\chi(0)$; and $\omega=0$ for all means.
7 **begin** *Computation step:*
8 $\mathcal{M}=$ComputationStep$(\mathfrak{M}, \chi, \kappa_i, \epsilon, n_\nu, \Lambda, n_e)$.
9 *(See Alg. 3 for details.)*
10 **begin** *Convergence step:*
11 **if** HasConverged$(\mathcal{C}, \mathcal{M})==$*`true`* **then** $cv=true$;
12 **else** $\mathcal{C}=\mathcal{M}$;
13 Increment the number of iterations : n_{it}++ (see Sec. 5).
14 **Return** $\mathcal{M}$

4.2.1 Epidemic Computation of the Encrypted Means

The epidemic algorithm for computing the encrypted means, called `EESum` in the following, consists in performing repeated point-to-point exchanges of the encrypted means of the current Diptych data structure. Each participant picks at random a contact in its local view with which it exchanges its own encrypted means. Both update their respective Diptych and can then start another exchange if the number of exchanges sufficient for the epidemic sum to converge (fixed beforehand) has not been reached. The local update rule is specifically designed to cope with additively-homomorphic encrypted data: any division of encrypted data is delayed until its decryption so that only additions and scalar multiplications are performed. Algorithm 2 gives the local update rule of the `EESum` algorithm.

4.2.2 Epidemic Noise Generation

The perturbation function consists in adding a controlled amount of noise to each encrypted mean before its decryption. It must necessarily be performed in a collaborative manner so that no single participant knows the exact value of noise (which

[8]Note that when a cluster is small, the perturbation of its (low) mean shall give an irrelevant value. As a side-effect, such an outlying mean will simply be ignored *de facto* in the next iterations (no time series will correspond to it). Thus, our experiments show the number of clusters varying according to these "lost" means (Section 6).

Algorithm 2: Local update rule over encrypted data

Input: The public encryption key χ, the encrypted current local value $\mathrm{E}_\chi(v_l)$, the current local weight ω_l, the current local number of exchanges performed so far n_l, the encrypted current remote value $\mathrm{E}_\chi(v_r)$, the current remote weight ω_r, the current remote number of exchanges performed so far n_r.

```
if n_r ≠ n_l then One value has to be scaled to be added
    if n_r > n_l; then Local value that must be scaled
        E_χ(v_l) = E_χ(v_l) ×_h 2^(n_r − n_l); ω_l = ω_l ×2^(n_r − n_l);
    else Remote value must be scaled
        E_χ(v_r) = E_χ(v_r) ×_h 2^(n_l − n_r); ω_r = ω_r ×2^(n_l − n_r);
E_χ(v_l) = E_χ(v_l) +_h E_χ(v_r); n_l = MAX(n_l, n_r) +1;
```

could otherwise be subtracted from the decrypted mean). In our context, the two parts of each encrypted mean must be perturbed, *i.e.*, its sum (a time-series) and its count (a real value). The epidemic perturbation function essentially amounts to the epidemic generation of $k*(n+1)$ Laplace random variables. It relies on two keystones: the Divisibility of Laplace lemma (see Lemma 1 in Section 3) and the `EESum` algorithm. Roughly, participants generate the noise-shares independently, as defined in Definition 5, and sum them up together through the `EESum` algorithm.

More precisely, each generated Laplace noise must satisfy two properties:

- **Correctness:** the noise must result from the sum of exactly n_ν distinct noise-shares (where n_ν is a parameter fixed beforehand);
- **Unicity:** the noise must be unique among the set of participants;

A straightforward value for the number of noise-shares is the size of the population, provided it is known beforehand. The noise generation is similar to the encrypted means computation: each participant generates a noise-share, encrypts it, and participates to the epidemic encrypted sum for approximating the global sum of the noise shares. However, the population size may not be known exactly in practice. The noise generation algorithm must be slightly adapted for this. First, the number of noise shares n_ν is set to an under-estimation of the population size, *e.g.,* to the estimated maximal lower bound. Thus, the epidemic noise generation will thus involve more noise shares than necessary: it can be corrected to involve the exact number of noise shares. Second, a cleartext counter, counting the number of nodes having actually participated in the noise generation, comes with the encrypted sum. A count being simply a sum of 1's, it follows the usual epidemic representation. The surplus of noise shares can now be estimated by subtracting n_ν to the actual value of the counter. Third, in order to guarantee the unicity property, *i.e.,* that all participants apply the same correction, each participant disseminates its locally generated correction, together with a random identifier. When participants disseminate their corrections, each one always keeps the correction associated to the smallest identifier. The probability of an incomplete dissemination is easily made negligible by standard dissemination protocols [23].

The two executions of the `EESum` algorithm - one for the encrypted means and one for their respective noise vectors - run in parallel. Their stopping criterion is the number of epidemic exchanges to perform for guaranteeing their convergence on the set of participants. After the sums, participants generate the noise corrections with their identifiers, and disseminate them, keeping the one with the smallest identifier at each exchange. The stopping criterion for the epidemic dissemination is also the number of exchanges necessary for convergence. Finally, when the dissemination terminates, each participant adds the corrected converged encrypted noises to the converged encrypted means to obtain the perturbed encrypted means and is now ready to launch their epidemic decryption.

Algorithm 3: Computation step

Input: The encrypted means $\mathfrak{M}$, the public encryption key χ, the private decryption key-share κ_i, the differential privacy level ϵ, the number of noise-shares n_ν, the local view Λ, the number of epidemic exchanges n_e necessary for the epidemic sum to converge (see Appendix B for fixing n_e).

```
begin Epidemic computation of encrypted means
    Background epidemic sums: Launch in background
    the encrypted epidemic algorithms for computing
    the encrypted means: 𝔐=EESum(𝔐, Λ, n_e, χ).
begin Epidemic noise generation
    Local noise-shares generation: Generate the vectors
    of noise-shares: 𝒩.ʃ and 𝒩.𝔠 where each noise-share ν_i =
    GenNoise(ϵ, n_ν). Set the epidemic counter ctr to 1.
    Epidemic noise computation: Launch in background
    the encrypted epidemic sum algorithms dedicated
    to noise-shares: 𝒩=EESum(𝒩, Λ, n_e, χ), and
    the epidemic count algorithm: ctr=EpiSum(Λ, n_e).
    Epidemic noise correction: If ctr > n_ν generate
    a correction vector cor = Σ_1^(ctr − n_ν) GenNoise(ϵ, n_ν)
    and a random identifier idcor for
    each of the noise vectors. Disseminate each epidemically
    cor=EpiDis(cor,idcor), and subtract each
    final correction vector to its respective noise vector.
    Encrypted perturbation: Add the
    vector of encrypted sums (resp. counts) to its vector of
    encrypted noises: 𝔐.ʃ = 𝔐.ʃ +_h 𝒩.ʃ and 𝔐.𝔠 = 𝔐.𝔠 +_h 𝒩.𝔠.
begin Epidemic decryption
    Local partial decryption: Choose a random local key-
    share identifier idk and decrypt partially 𝔐.ʃ and 𝔐.𝔠 with
    the local key-share: 𝒮=D_κi(𝔐.ʃ) and 𝒦=D_κi(𝔐.𝔠).
    Epidemic decryption: Launch
    the epidemic decryption: 𝒮=EEDec(𝔐.ʃ, 𝒮, idk,
    Λ, κ_i)/𝔐.ω and 𝒦=EEDec(𝔐.𝔠, 𝒦, idk, Λ, κ_i)/𝔐.ω.
Compute the vector of perturbed means: 𝓜 = 𝒮/𝒦.
Smooth the vector of perturbed means:
𝓜=Smooth(𝓜) (see Sec. 5).
Return 𝓜.
```

4.2.3 Epidemic Decryption

Similarly to the epidemic noise generation algorithm, epidemic decryption must be performed collaboratively, and be both correct and unique. First, collaboration is achieved based on the threshold decryption feature of the Damgard-Jurik encryption scheme (1) by assigning one key-share to each participant (*e.g.,* when it acquires the parameters) and (2) by decrypting partially the means at each epidemic decryption exchange. Second, unicity is trivially enforced through the convergence properties of the epidemic sum algorithm. Indeed, both the encrypted means and the encrypted noise vectors are guaranteed to be unique among the set of participants. Participants can decrypt them concurrently without threatening the unicity property. Third, correctness requires to apply exactly t distinct key-shares to

the encrypted means of each participant. To this end, each participant is equipped with (1) a random *key-share identifier* and (2) a set storing the identifiers of the key-shares that have partially decrypted its current means. During an epidemic decryption exchange, both participants check that their respective set of identifiers does not already contain that of the other. Additionally, decryption latency can be reduced by erasing and replacing the partially decrypted means of the less advanced participant with that of the more advanced participant. The stopping criterion of the epidemic decryption algorithm is the equality between the cardinality of the set of key-shares applied and the required number of key-shares.

4.2.4 Termination

The traditional centralized termination criterion of k-means is centroids convergence. However, the convergence threshold is an arbitrary parameter that fixes a limit on the distance between two consecutive sets of centroids. Convergence becomes hard to guarantee when centroids are perturbed to satisfy differential privacy. In order to still guarantee termination (and to avoid exceeding the privacy parameters ϵ and δ), we simply limit the number of iterations to be executed. The termination criterion is thus the disjunction between the convergence of centroids and a limit on the number of iterations (denoted by n_{it}^{max} in the following). Section 5 discusses strategies dedicated to the production of high-quality centroids in a reduced number of iterations as well as the use of a smarter termination criterion that depends on the evolution of the centroids quality along iterations.

4.3 Correctness and Security

Theorem 1 states the correctness of Chiaroscuro and Theorem 2 states its security. We refer the interested reader to the Appendix for the proofs.

THEOREM 1. *Any instance π of Chiaroscuro (Algorithm 1) executed on a set of honest-but-curious participants is correct.*

THEOREM 2. *Any instance π of Chiaroscuro (Algorithm 1) executed on a set of honest-but-curious participants is secure.*

4.4 Extensions to Malicious Attackers

Chiaroscuro can be extended to cope with *malicious* attackers that deviate from the execution sequence. First, the security of participants can be increased by benefiting from the current rise of highly secure *trusted execution environments* embedded into personal devices (such as, *e.g.,* in [2]) so that tampering the execution sequence, which amounts to tampering the code executed, becomes prohibitively costly. Note that data integrity can also be enforced by these environments, hence countering "lying" malicious participants. Second, the population of participants can be strengthened by restricting the access to the execution sequence to the set of authorized devices through usual authentication techniques. Finally, the collaborative nature of the execution sequence (transparency requirement R1) is a sound basis for detecting malicious attacks thanks to the systematic dissemination of malicious behaviors (*e.g.,* checking that decrypted values are all equal across participants (epidemic dissemination)).

5. CLUSTERING QUALITY

Chiaroscuro discloses the centroids produced by each k-means iteration in order to achieve both scalability and fault-tolerance, and protects them through an appropriate, differentially private, perturbation scheme. However, the injection of noise in each intermediate result of the k-means algorithm has an impact on the quality of the final centroids. Naive strategies may even totally thwart the algorithm by injecting an overwhelming amount of noise. This section is dedicated to maintaining the quality of the clustering with perturbation, to a level comparable to the level reached by clustering with no perturbation, which corresponds to requirement R3. Two levels can be delved into for reducing the impact of the perturbation: the quality of the sequence of centroids (Section 5.1) and the quality of each centroid (Section 5.2).

5.1 Budget Concentration Strategies

A naive privacy budget distribution strategy could consist roughly in estimating the number n_{it} of iterations needed for a centralized k-means to converge, setting the maximum number of iterations to n_{it}, and assigning a budget equal to ϵ/n_{it} to each iteration. However, the estimated number of iterations may be high depending on the dataset, the k-means convergence threshold, and the initial centroids. Hence, a relevant solution is to concentrate the budget on the first iterations. The k-means algorithm is well-known for its logarithmic error loss rate [4]: the highest gains in quality are obtained during the first iterations.

We propose three proofs-of-concept budget concentration strategies that strive to preserve the quality of the centroids produced during the first iterations:

- **GREEDY (G):** The GREEDY strategy (G for short) favors the first iterations through an *exponential decrease* in the budget assignment: $1/2^i$ of the budget is assigned to the i^{th} iteration. Since $\sum 1/2^i$ is bounded by 1, the privacy budget finally spent never exceeds ϵ;
- **GREEDY_FLOOR (GF):** The GREEDY_FLOOR strategy (GF for short) spreads each GREEDY assignment on a floor of f iterations. Each of the first f iterations is assigned $1/(2f)$ of the budget, each of the second f, $1/(2^2 f)$, and so forth. The budget assignment is thus *exponential by floor*;
- **UNIFORM_FAST (UF):** The UNIFORM_FAST strategy (UF for short) bounds the number of iterations to a strong limit (e.g., $n_{it} = 5$) and distributes the budget uniformly among them. The budget assignment is thus *constant* but limited to a reduced number of iterations;

The magnitude of the noise added by G and GF strategies increases with the number of iterations, possibly leading to a noise overwhelming the centroids before termination, and consequently to their non-convergence. Nevertheless, termination is still guaranteed by the limit on the number of iterations. (It is easy to make the termination criterion smarter by identifying the moment at which the noise becomes intractable and stop at that moment[9]. For simplicity, this work focuses on the criterion described above.) Other strategies can be designed, experimented, and finally plugged into Chiaroscuro if they lead to better results.

5.2 Smoothing the Means

After being perturbed, a mean $\mathcal{M}[i]$ might be far from its value prior to perturbation. In this case, data assigned to $\zeta[i]$ will probably be re-assigned to $\zeta[j]$ where $j \neq i$, which is against the convergence criterion of k-means. However, the noise added to each

[9]Participants can monitor the centroids quality through the inter-cluster inertia (Definition 1): (1) the cardinality of each cluster is already computed during the means computation and (2) the center of mass of the full dataset and its cardinality can be easily computed based on the encrypted gossip sum algorithm and the distributed noise generation, once and for all, before launching Chiaroscuro. Chiaroscuro would then stop when the quality starts to drop.

Dataset	
Number of time-series	3M (CER), 1.2M (NUMED)
Size of time-series	24 (CER), 20 (NUMED)
Privacy	
Key size	1024 bits
Key-shares threshold	$\tau \in [0.001\%,10\%]$
Privacy budget	$\epsilon = 0.69$
Nb of noise-shares	$n_\nu = 100\%$
k-means	
Initial nb of centroids	$k = 50$
GOSSIP	
Size of the local view	30
Churn	From 10% to 50%
Quality	
Floor size (G)	4
Max nb of iterations	$n_{it}^{max} = 5$ (UF only), $n_{it}^{max} = 10$
Moving average (SMA)	20%

Table 2: Experimental Parameters

measure is a Laplace random variable: it can be positive or negative with equal probability. Therefore, given a sliding window of size $w+1$, we can smooth the values of the sum part of $\mathcal{M}[i]$, *i.e.,* $\mathcal{S}[i]$, by averaging them as follows. Let $\mathcal{S}[i] = <\mathcal{S}[i,1]...\mathcal{S}[i,n]>$ be the sum part after perturbation and $\overline{\mathcal{S}[i,j]}$ be the smoothed value of $\mathcal{S}[i,j]$. Let $\mathtt{m}(\mathcal{S}[i,j])$ be the j^{th} (modulo n) value in $\mathcal{S}[i]$, then $\overline{\mathcal{S}[i,j]} = (\mathtt{m}(\mathcal{S}[i,j-w/2]) + ... + \mathtt{m}(\mathcal{S}[i,j+w/2]))/(w+1)$. After having been smoothed by this *simple moving average* technique (denoted by SMA hereafter), the new mean is ready for the next iteration, guaranteeing both privacy (the output of a function performed on a differentially private input is differentially private too) and a better stability of k-means.

6. EXPERIMENTAL VALIDATION

6.1 Settings

We have implemented Chiaroscuro's execution sequence and integrated it into *Peersim* [30], a well-known distributed computing simulator. This implementation is a strong proof of concept and allows us to run experiments over medium-sized populations (*e.g.,* tenths of thousands of participants) on a single machine. However, experiments involving very large scale populations (*e.g.,* several millions of participants) would need a testing platform deployable over a number of nodes sufficient for coping with the load. To the best of our knowledge, such a platform is not available today. As a result, in order to launch experiments on very large scale populations, we evaluate: (1) the local costs (*i.e.,* CPU times and bandwidth consumption) by performing encryptions, decryptions, and encrypted additions on a typical participant, (2) the quality (*i.e.,* intra-cluster inertia) by running a *perturbed* centralized k-means implementation embedding our budget concentration strategies and means smoothing technique, and (3) the latency (*i.e.,* average number of messages per participant) by simulating the epidemic sum, dissemination, and decryption algorithms. Moreover, because of their arbitrary connection patterns, participants may join or leave an ongoing run of Chiaroscuro at any moment of its execution. The resulting churn impacts the quality of centroids (1) by making unavailable a random fraction of the population at each gossip exchange and (2) by dynamically changing the input set of time-series at each k-means iteration. We also evaluate these two aspects below with a fixed latency. The parameters values we used are synthesized in Table 2. They consist in typical values met in practical contexts.

6.1.1 Experimental Platform

In real-life, Chiaroscuro would be executed on a variety of desktops, tablets, smartphones, or laptops. All are equipped today with GB RAM and multi-core CPU. The experiments have been run on such hardware: a laptop containing an eight-core 2.6 GHz CPU with 8 GB RAM. The programming language was Java 7.

We evaluate Chiaroscuro with respect to a real dataset and a synthetic one, each related to a targeted application domain:

- **CER** The CER dataset [16] contains the electricity consumption time-series of thousands of Irish homes and businesses collected between 2009 and 2010 at the rate of one sample every half hour. In our experiments, we use three millions daily time-series randomly sampled from the dataset and reduced to twenty-four measures each (electricity consumption per hour). The range of each measure is [0,80] (sum sensitivity thus equals to 1920).
- **NUMED** The NUMED dataset is a set of 1.2 million time-series representing the tumor growth of cancer suffering patients. It was synthetically generated based on mathematical models of typical profiles [7] (at this scale, real health-related time-series are hard to access), at the rate of one sample every week, and for twenty weeks per patient. Each time-series contains one measure per week taking its value in [0,50] (sum sensitivity thus equal to 1000).

6.1.2 Settings: Local Costs

Three parameters fully determine the bandwidth consumption and the computation times of a participant. First, the number of clusters, initially set to $k = 50$. Second, the size of a mean, equal to the size of a time-series (*i.e.,* twenty-four measures for the CER dataset and twenty measures for the NUMED dataset). Third, the length (in bits) of an encrypted value in a mean. It is similar to the size of the cryptographic key, which we set to 1024 bits in our experiments (average security). The linear relationship between these parameters and the local costs is simple, making easy to estimate the costs corresponding to other values for these parameters.

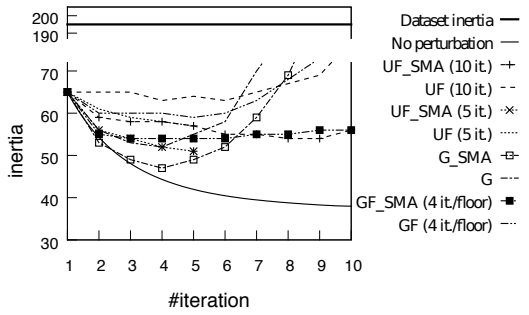
6.1.3 Settings: Quality

The perturbed centralized k-means satisfies (ϵ, δ)-probabilistic differential privacy where ϵ is set to a common value [11]: $\epsilon = \ln 2 = 0.69$. Since the value of δ has no impact on the centroids quality, we can simply ignore it here[10]. The floor-size of the GREEDY_FLOOR strategy was set to 4. We set the initial number of clusters to $k = 50$. For the synthetic NUMED dataset, the initial centroids are chosen uniformly at random within the set of (synthetic) time-series. For the CER dataset, they are generated by CourboGen [9], the EDF bigdata generator (obvious privacy reasons preclude raw real-life time-series to be used as initial centroids).

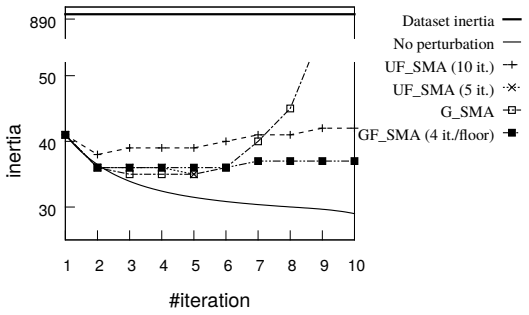
6.1.4 Settings : Latency

The execution sequence of Chiaroscuro is made of two epidemic encrypted sums and one epidemic decryption. We evaluate experimentally the average number of messages per participant (1) to compute local approximations of the global sum, and (2) to decrypt the encrypted perturbed means. In order to exhibit the tendency of each curve, we vary the size of the

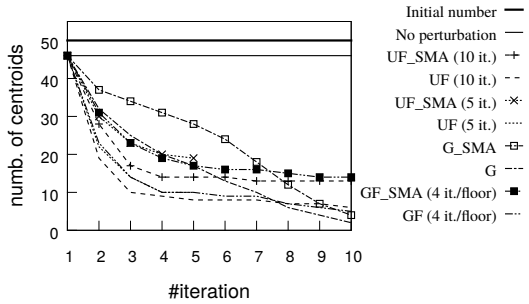
[10]In practice the δ parameter is set to a value close to 1. The larger the value of δ, the larger the number of gossip exchanges. Theorem 3 (from [25] - see Appendix B) shows that with the current connectivity layer of Chiaroscuro, high δ values (*e.g.*, $\delta = 0.995$) can be reached with a reasonable number of gossip exchanges (*e.g.*, $n_e = 47$).



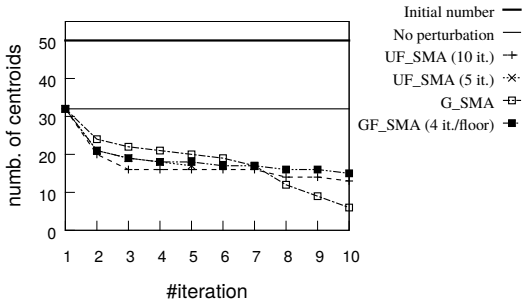
(a) CER: Evolution of the Pre-Perturbation Intra-Cluster Inertia



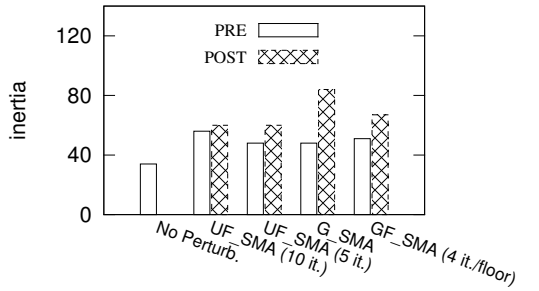
(b) NUMED: Evolution of the Pre-Perturbation Intra-Cluster Inertia



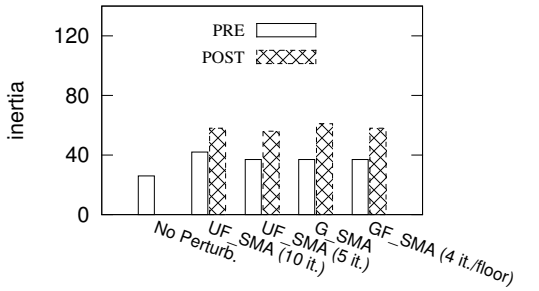
(c) CER: Evolution of the Number of Centroids



(d) NUMED: Evolution of the Number of Centroids

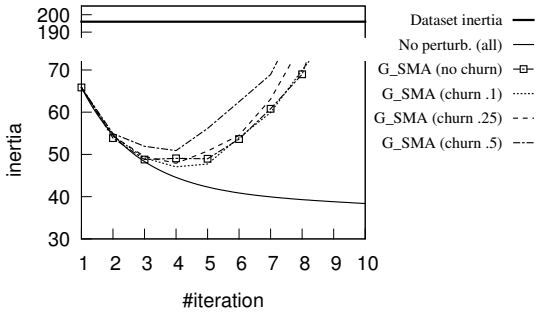


(e) CER: (1) Lowest Pre-Perturbation Inertia (PRE) and (2) Corresponding Post-Perturbation Inertia without Re-Assignment (POST)

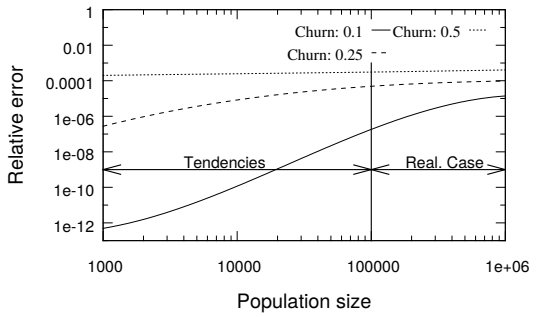


(f) NUMED: (1) Lowest Pre-Perturbation Inertia (PRE), (2) Corresponding Post-Perturbation Inertia without Re-Assignment (POST)

Figure 2: Quality of the Perturbed Clustering - CER: $3M$ **time-series and NUMED :** $1.2M$ **time-series**

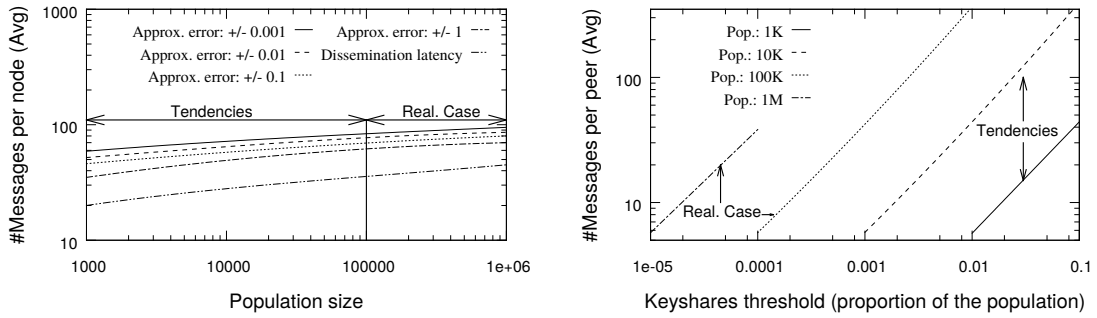


(a) *Churn-Enabled:* Evolution of the Pre-Perturbation Intra-Cluster Inertia (CER ($3M$ time-series))



(b) *Churn-Enabled:* Relative Error between the Result of the Epidemic Encrypted Sum and the Exact Value of the Sum (100 Messages per Participant on Average)

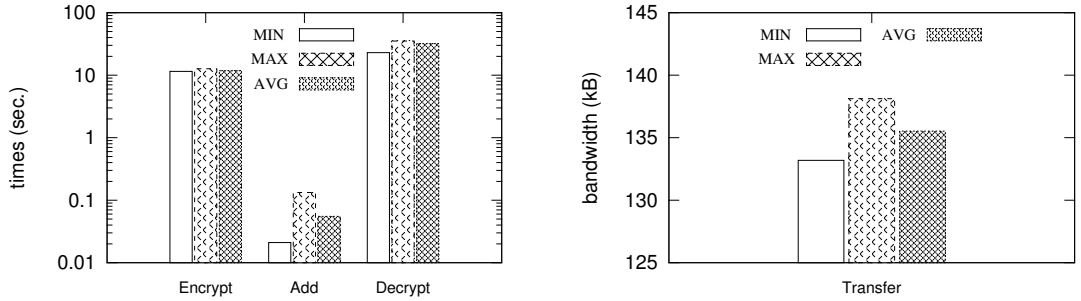
Figure 3: Impact of Churn



(a) Epidemic Encrypted Sum (Means Computation and Noise Generation) and Epidemic Dissemination (Noise Correction)

(b) Epidemic Decryption

Figure 4: Internal Latencies of the Computation Step



(a) Times Consumption for Encrypting or Decrypting One set of Means, or Adding Together Two Sets of Means (seconds)

(b) Bandwidth Consumption for Transferring One Set of Means (kilo-bytes)

Figure 5: Unitary Local Costs for a Set of 50 Means, 20 Measures per Mean, and a 1024 Bits Encryption Key

population from $n_p = 1K$ participants to $n_p = 1M$. We set the size of the local view per participant to $n_\Lambda = 30$ participants (an average case [21]). For the encrypted epidemic sum, we set the local data of any participant to a single integer value equal to 1, and launch the experiments for several absolute approximation errors targeted, *i.e.,* from 0.001 to 1. Concerning the epidemic decryption, this is the number of key-shares needed for decryption that we vary, expressed as a fraction of the population size, *i.e.,* from $\tau = 0.001\%$ to $\tau = 10\%$.

6.1.5 Settings : Impact of Churn on Quality

We model the churn as the uniform probability for each participant to be disconnected (1) at each gossip exchange of the epidemic encrypted sum and (2) at each perturbed k-means iteration. For the former, we measure the relative error of the encrypted epidemic sum for a number of participants varying from $n_p = 1K$ to $n_p = 1M$, a number of messages per participant equal to 100 on average, all participants having as local data a single integer equal to 1, and for three disconnection probabilities (at each gossip exchange) of 10%, 25%, and 50%. For the later, we measure the evolution of the intra-cluster inertia during the perturbed k-means clustering of the CER dataset in the same setting with no-churn one and with the above three disconnection probabilities (at each iteration).

6.2 Clustering Quality

We evaluate the impact of the perturbation on the clustering quality by monitoring the evolutions along the iterations of the k-means algorithm of (1) the intra-cluster inertia (both prior to perturbation and after) (2) the number of centroids. We run each experiment ten times and plot the average values. For keeping the figures readable, we do not plot the confidence intervals but give them in the corresponding paragraph.

Figure 2(a) shows the evolution of the intra-cluster inertia along the first ten iterations of the k-means algorithm executed over the CER dataset. We plot (1) the full inertia of the dataset (a constant, the *worstcase upper bound* of the intra-cluster inertia), (2) the evolution of the intra-cluster inertia in the traditional unperturbed k-means (a decreasing logarithm, the *best-case lower bound*), and (3) the evolution of the intra-cluster inertia in the perturbed k-means before perturbing the centroids. Each budget concentration strategy is shown, both with the means smoothing and without. Globally, the inertia in the perturbed cases stay close to the lower bound, especially during the first iterations where the k-means error loss rate is at its highest. The strategy that reaches the best inertia level is GREEDY (third to fifth iteration). For all strategies, the SMA smoothing positively impacts inertia. Figure 2(c) also shows that GREEDY with SMA smoothing preserves the highest number of centroids. This high number of centroids further explains partially the difference between its pre-perturbation inertia and its post-perturbation inertia shown in Figure 2(e) (where aberrant centroids unable to cope with the noise, as explained in Section 4.2, have been removed). Similar conclusions can be drawn with the NUMED dataset (Figures 2(b), 2(d), and 2(f)). We do not plot the inertia reached without smoothing because they are very close to the inertia reached with it. This can be explained by the distribution of data: the NUMED time-series are equally distributed across the clusters (contrary to the strongly concentrated CER time-series), keeping low the number of small clusters, *i.e.,* those that benefit the most from smoothing

because of their high sensitivity to noise. Whatever the dataset and budget distribution strategy, the confidence intervals of the evolution of the inertia and number of centroids are similar, small, and stable during the first iterations (*i.e.*, ± 2.5 points (resp. 1) for the CER (resp. NUMED) inertia, and ± 2 points (resp. 1) for the CER (resp. NUMED) number of centroids) but they diverge when the noise starts to overwhelm centroids.

6.3 Performance

6.3.1 Local Bandwidth and Time Consumption

Figure 5 shows the minimum, maximum, and average time and bandwidth consumption per participant for a Peersim execution of Chiaroscuro with 20 measures per mean. Figure 5(a) shows the time required for encrypting a set of means, decrypting it, and adding together two sets of encrypted means. The decryption time represents the highest cost. However, being executed only once during an iteration and staying below a few seconds in our experiments, it remains affordable when executed in background by the participant. The addition of two sets of encrypted means is the most frequent operation. The time it consumes is below 0.1 second, *i.e.,* two orders of magnitude lower than the decryption times. Figure 5(b) shows the bandwidth consumption, in kilo-bytes, for transferring a set of encrypted means. A hundredth of kilo-bytes per transfer are necessary, which amounts to around 1 second for a humble 1 Mb/s connection speed. The epidemic sum algorithm requires two encrypted means transfers and the *once per iteration* epidemic decryption algorithm transfers both the encrypted means and their partially decrypted version - the equivalent of four encrypted means. Both algorithms thus transfer a few hundredth of kilo-bytes per epidemic exchange, which is clearly affordable given today's bandwidths.

6.3.2 Global Latency

Figure 4(a) depicts the number of messages for performing an encrypted epidemic sum given absolute approximation errors varying from ± 0.001 to ± 1. The number of messages remains very low, *i.e.*, under the hundred, even for a million participants population with the tightest absolute approximation error of ± 0.001 (*i.e.*, equal to 10^{-9} the exact sum value)[11]. Moreover, the figure clearly shows that the number of messages grows logarithmically depending on the population size. On the same figure, we plot the dissemination latency of the smallest identifier noise correction. It remains in the same order of magnitude, and depends logarithmically too on the population size. Figure 4(b) shows the average number of messages per participant for performing epidemic decryption with respect to the key-shares threshold expressed as a fraction of the population size. The decryption latency exhibits a linear behavior depending on the size of the threshold. For one million participants, our experiments reach the limit of the experimental platform at a threshold size greater than 0.01% the population size. However, the linearity of the curves lets forecast a number of messages on the order of magnitude of the hundred for a realistic threshold size of 0.01% (*i.e.,* 100 distinct participants to contact for decryption). Finally, the total latency of an iteration is the latency of two epidemic encrypted sums, one epidemic dissemination, and one epidemic decryption. Given the practical settings discussed above (*i.e.,* one million participants, an approximation error in the sum of ± 0.001, and a threshold size of 0.01%), the total latency of an iteration is on the order of a few hundreds of messages, dominated by the epidemic encrypted sums. Injecting the local costs observed above (NUMED dataset, G_SMA strategy) results in a first iteration completing after around 26 mins and a fifth one after around 10 mins (the effect of the noise made aberrant 60% of the initial number of centroids between the first and the fifth iteration). These times fit the context targeted by Chiaroscuro, where the users are the individuals and the results do not suffer from severe time constraints (*e.g.*, background analysis).

[11] With $n_e = 100$ gossip exchanges and a maximal approximation error $e_{max} = 10^{-9}$ (the other parameters reamining the same as above) and according to Theorem 3 (from [25] - see Appendix B), $\delta = (1 - 2 \cdot 10^{-51})^{480}$, which is very close to 1.

6.4 Impact of Churn on Quality

The churn-enabled experiments (Figures 3(a) and 3(b)) show a low sensitivity to churn. First, the churn-enabled intra-cluster inertia follow closely those without churn. Indeed, the exclusion of a random fraction of the highly redundant CER time-series at each iteration does not incur strong skews. The increasing divergence of the last iterations suggests that the reduction of the cardinalities of clusters is of higher importance : the higher the churn, the smaller the clusters and consequently the less robust to a high-magnitude noise. Second, the relative error of the encrypted epidemic sum due to the churn (Figure 3(b)) represents a negligible fraction of the exact sum for the three levels of churn (at most a bit less than 0.1% for the highest disconnection rate of 50% at each gossip exchange).

7. RELATED WORK

Generic secure multi-party computation (SMC) techniques [15, 37] are able to produce a distributed and secure version of any centralized polynomial-time algorithm. However these techniques cannot be used in practice for non-trivial algorithms and non-tiny input data because of the prohibitive cost of the distributed versions produced.

Specific SMC algorithms address the problem of clustering horizontally partitioned datasets [5, 17, 18, 19, 20, 22, 26, 27, 35]. Most of them consider only two participants, choose k-means or a similar algorithm for performing the clustering, and all tackle the honest-but-curious attack model. In [22], the algorithm is made of a local phase computing the local means, and a global phase averaging the two local sets of means. The computation of the means taking place at each iteration is protected through an homomorphic encryption scheme. However, the means are disclosed at the end of each iteration, a breach that can lead to uncontrolled data leaks. The authors of [20] distribute the k-means algorithm by splitting each centroid in two complementary random parts, one per participant. A sequence of SMC algorithms implements the usual k-means steps, guaranteeing a strong security level. However, the use of generic SMC techniques raises strong concerns about fault-tolerance and scalability. The solution proposed in [5] is also based on random parts and SMC algorithms, but it avoids generic SMC techniques through a smart use of homomorphic encryption and of a secure scalar product algorithm [13]. However, the achievements of scalability and fault-tolerance are still uncertain (transfer of the full database of a participant, encrypted, to the other participant, and use of random parts and SMC algorithms). The authors of [19] (which extends [17] and [18]) use similar SMC algorithms as well as random parts, and inherit from their drawbacks. The authors of [35] consider many more participants but (1) organize them in a rigid tree-like structure and use random parts and generic SMC techniques, which questions fault-tolerance and scalability, and (2) disclose the raw means, which raises security concerns. Other work has considered the adaptation of other clustering algorithms (*e.g.,* EM in [26], DBSCAN in [27]).

However, they use similar building blocks (random parts and SMC algorithms) and thus suffer from similar drawbacks.

Finally, the problem of computing differentially-private aggregates in a distributed manner has attracted attention in the last decade (*e.g.,* [1, 6, 33, 36]). This line of work usually (1) assumes the presence of at least one central entity, notably for driving the execution of (2) simple aggregate queries over (3) thousands of participants. On the contrary, our work (1) considers a full iterative k-means execution sequence (both specific and complex) and (2) fully distributes it (high parallelism, no central bottleneck, no unnecessary additional security/resource assumption) over (3) millions of participants.

8. CONCLUSION

We proposed Chiaroscuro, the first solution to address the problem of clustering massively distributed personal data with sound privacy guarantees. Our original Diptych data structure allows adapting k-means to a computing environment made of autonomous personal computing devices. Our gossip-based *modus operandi* provides scalability and fault-tolerance, while a novel combination of homomorphic encryption with differential privacy shields it against colluding dishonest nodes. We showed through an extensive experimental validation that Chiaroscuro achieves a quality comparable to centralized k-means executions at an affordable cost. Exciting research perspectives include characterizing the algorithms that Chiaroscuro can support. The class of iterative analytical algorithms (*e.g.,* expectation-maximization or probabilistic matrix factorization) especially fits the foundations laid down by Chiaroscuro.

9. REFERENCES

[1] G. Ács and C. Castelluccia. I have a dream!: Differentially private smart metering. In *IH*, pages 118–132, 2011.

[2] T. Allard, N. Anciaux, L. Bouganim, Y. Guo, L. Le Folgoc, B. Nguyen, P. Pucheral, I. Ray, I. Ray, and S. Yin. Secure personal data servers: A vision paper. *VLDB*, 3(1-2):25–35, 2010.

[3] O. T. Alliance. *Data Protection & Breach*, 2014.

[4] L. Bottou and Y. Bengio. Convergence properties of the kmeans algorithm. In *ANIPS*, volume 7. Denver, 1995.

[5] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *CCS*, pages 486–497, 2007.

[6] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards statistical queries over distributed private user data. In *NSDI*, pages 169–182, 2012.

[7] L. Claret, M. Gupta, K. Han, A. Joshi, N. Sarapa, J. He, B. Powell, and R. Bruno. Evaluation of tumor-size response metrics to predict overall survival in western and chinese patients with first-line metastatic colorectal cancer. *J. Clin. Onc.*, 31(17):2110–2114, 2013.

[8] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC*, pages 119–136, 2001.

[9] L. D. P. dos Santos, A. G. da Silva, B. Jacquin, M.-L. Picard, D. Worms, and C. Bernard. Massive smart meter data storage and processing on top of hadoop. In *Int. Work. on End-to-end Man. of Big Data, VLDB*, 2012.

[10] C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.

[11] C. Dwork. A firm foundation for private data analysis. *CACM*, 54(1):86–95, 2011.

[12] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[13] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *ICISC*, pages 104–120, 2004.

[14] O. Goldreich. Foundations of cryptography: a primer. *Found. Trends in Theoretical Computer Science*, 1(1):1–116, April 2005.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *STOC*, pages 218–229, 1987.

[16] ISSDA. *The Commission for Energy Regulation, Electricity Customer Behaviour Trial*, 2012. http://www.ucd.ie/issda.

[17] G. Jagannathan, K. Pillaipakkamnatt, and D. Umano. A secure clustering algorithm for distributed data streams. In *ICDM Work.*, pages 705–710, 2007.

[18] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright. A new privacy-preserving distributed k-clustering algorithm. In *SDM*, pages 494–498, 2006.

[19] G. Jagannathan, K. Pillaipakkamnatt, R. N. Wright, and D. Umano. Communication-efficient privacy-preserving clustering. *Trans. Data Privacy*, 3(1):1–25, 2010.

[20] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *SIGKDD*, pages 593–599, 2005.

[21] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *Trans. Comp. Sys.*, 23(3):219–252, Aug. 2005.

[22] S. Jha, L. Kruger, and P. McDaniel. Privacy preserving clustering. In *ESORICS*, pages 397–417, 2005.

[23] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS*, pages 482–491, 2003.

[24] S. Kotz, T. J. Kozubowski, and K. Podgorski. *The Laplace Distribution and Generalizations*. 2001.

[25] W. Kowalczyk and N. A. Vlassis. Newscast EM. In *NIPS*, pages 713–720, 2004.

[26] X. Lin, C. Clifton, and M. Zhu. Privacy-preserving clustering with distributed EM mixture modeling. *Know. Inf. Sys.*, 8(1):68–81, 2005.

[27] J. Liu, J. Z. Huang, J. Luo, and L. Xiong. Privacy preserving distributed DBSCAN clustering. In *EDBT-ICDT Work.*, pages 177–185, 2012.

[28] S. Lloyd. Least squares quantization in PCM. *Trans. Inf. Theor.*, 28(2):129–137, 1982.

[29] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *ICDE*, pages 277–286, 2008.

[30] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *P2P*, pages 99–100, 2009.

[31] M. Newborough and P. Augood. Demand-side management opportunities for the uk domestic sector. *Gen., Trans. and Dist.*, 146(3):283–293, 1999.

[32] A. Prudenzi. A neuron nets based procedure for identifying domestic appliances pattern-of-use from energy recordings at meter panel. In *PESW*, volume 2, pages 941–946, 2002.

[33] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, pages 735–746, 2010.

[34] I. Risk Based Security. *Data Breach QuickView*, 2014.

[35] J. Sakuma and S. Kobayashi. Large-scale k-means clustering with user-centric privacy preservation. In *PAKDD*, pages 320–332, 2008.

[36] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.

[37] A. C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

APPENDIX

A. ACKNOWLEDGEMENTS

The authors warmly thank Patrick Valduriez for his thorough proofreadings as well as the anonymous reviewers for their detailed feedback.

B. SECURITY

B.1 Differential Privacy of the Perturbation Mechanism

The encrypted sum and count vectors of the means must be perturbed by vectors of random variables that are sampled in the appropriate Laplace distribution in order to satisfy differential privacy. In Chiaroscuro, the Laplace noises are computed based on the `EESum` algorithm and the infinite divisibility of the Laplace distribution (Lemma 1): the `EESum` algorithm simply performs the homomorphically-encrypted sum of noise-shares (Definition 5) generated locally by each participant. Such gossip-based noise computation differs from a standard centralized noise generation algorithm in two ways. First, gossip aggregation algorithms are approximate in nature; the resulting noise thus contains an approximation error. Second, due to the massive distribution of gossip algorithms, the number of noise-shares involved in the noise computation may be hard to know and to fix beforehand. This may result in an additional error in the noise generated. We show below that probabilistic differential privacy is satisfied.

B.1.1 Approximate Sum

Although gossip aggregation algorithms are inherently approximate, their approximation errors can be made arbitrarily small in practice. Indeed, the gossip approximation error depends essentially on the number of gossip exchanges per participant. First, it is guaranteed to converge to zero exponentially fast [21, 23] (see Theorem 3 below). Second, since the number of exchanges per participant is a parameter, it can be set so that the approximation error is as small as desired - *i.e.*, typically several orders of magnitude lower than the exact aggregate value. We show below how the approximation error due to the `EESum` algorithm is taken into account in the achievement of the (ϵ, δ)-probabilistic differential privacy model. First, Lemma 2 shows how to compensate the approximation error. Then, Theorem 3 (from [25]) shows how to compute the minimum number of gossip exchanges per participant required for reaching the desired (ϵ, δ) privacy level.

LEMMA 2. *Let S denote the gossip approximate sum of values, i.e., $S=(1+e_S)\cdot\sum_i x_i$ where $x_i\in[d_{min},d_{max}]$ and e_S denote the relative gossip approximation error. The approximation error is such that $0<|e_S|\leq e_{max}$ with probability at least $1-\iota$ [23]. Let $\lambda=(1+e_{max})\cdot\max(|d_{min}|,|d_{max}|)/\epsilon$. Let N denote the gossip approximate sum of noise-shares, i.e., $N=(1+e_N)\cdot\sum_i^{n_\nu}\nu_i=(1+e_N)\cdot\mathcal{L}(\lambda)$ where n_ν is the number of noise-shares, each ν_i is a noise-share (Definition 5), and e_N denote the relative gossip approximation error. Similarly to e_S, e_N is such that $0<|e_N|\leq e_{max}$ with probability at least $1-\iota$. Then, $S'=S+\overline{N}$ satisfies (ϵ, δ)-probabilistic differential privacy, where $\overline{N}=(1+e_{max}/(1-e_{max}))\cdot N$ and $\delta=(1-\iota)^2$.*

PROOF. The sensitivity of S is equal to : $(1+e_{max})\cdot\max(|d_{min}|,|d_{max}|)$ with probability at least $1-\iota$. Adding to S a perturbation sampled in $\mathcal{L}(\lambda)$, where $\lambda=(1+e_{max})\cdot\max(|d_{min}|,|d_{max}|)/\epsilon$, would thus satisfy ϵ-probabilistic differential privacy with probability at least $1-\iota$ by definition. However, the gossip sum of noise-shares does not yield the exact perturbation because of its approximation error : $N=(1+e_N)\cdot\mathcal{L}(\lambda)$ where $0<|e_N|\leq e_{max}$ with probability at least $1-\iota$. Since e_N may be negative, the magnitude of the perturbation may be reduced, which may question the privacy guarantees. The maximum loss occurs when e_N is minimum, *i.e.,* when $e_N=-e_{max}$. It can be compensated by slightly increasing the perturbation. Let c denote the relative compensation. The objective is to have $(1+c)\cdot N\geq\mathcal{L}(\lambda)$, so $(1+c)\cdot(1+e_N)\cdot\mathcal{L}(\lambda)\geq\mathcal{L}(\lambda)$. At worst, $e_N=-e_{max}$, so we have $(1+c)\cdot(1+e_N)\cdot\mathcal{L}(\lambda)\geq(1+c)\cdot(1-e_{max})\cdot\mathcal{L}(\lambda)$. As a result, after the straightforward development of $(1+c)\cdot(1-e_{max})\cdot\mathcal{L}(\lambda)\geq\mathcal{L}(\lambda)$, we must have $c\geq e_{max}/(1-e_{max})$. So the approximation error of the gossip sum of noise-shares can be compensated by increasing the perturbation by a factor equal to $c=e_{max}/(1-e_{max})$. Consequently, $S'=S+(1+e_{max}/(1-e_{max}))\cdot(1+e_N)\cdot\mathcal{L}(\lambda)$ satisfies (ϵ, δ)-probabilistic differential privacy, where $\lambda=(1+e_{max})\cdot\max(|d_{min}|,|d_{max}|)/\epsilon$ and $\delta=(1-\iota)^2$. □

Next, we show how to compute the minimum number of gossip exchanges required. The exact convergence speed depends on the underlying network topology. The current version of Chiaroscuro relies on Newscast [25] for managing the connectivity between participants. The convergence speed of aggregate computation in Newscast is given by Theorem 3 [25], which allows to compute the minimum number of gossip exchanges required for reaching the desired approximation error with the desired probability. If Chiaroscuro changes its connectivity layer, the same computation would apply based on a different formula. For simplicity, Theorem 3 considers a synchronous setting where each participant performs a gossip exchange one after the other.

THEOREM 3 (FROM [25]). *With probability $1-\iota$, after $n_e=\lceil 0.581(\log n_p+2\log s+2\log\frac{1}{e_{max}}+\log\frac{1}{\iota})\rceil$ exchanges per participant, we have $\max_i|\sigma_i-\sigma|\leq e_{max}$, where σ_i is the estimate of the sum component of individual i's local state and σ is its exact global value, n_p is the cardinality of the population, $e_{max}>0$ is an upper bound on the approximation error, and s^2 is the data variance.*

As a result, given the targeted privacy parameter δ and the maximum number of iterations n_{it}^{max}, the differential privacy probability assigned to each perturbed value is computed : $\delta_{atom}=\sqrt[(n_{it}^{max}*2n)]{\delta}$. Then, given δ_{atom}, the maximal error e_{max}, and the expected data variance s^2 (if unknown, it can be set to, *e.g.*, an estimated upper bound), the minimum required number of gossip exchanges n_e per execution of the `EESum` algorithm is computed (Theorem 3). For example, with $\delta=0.995$, $e_{max}=10^{-12}$, $s^2=1$, and values from Section 6 (*i.e.*, $n_{it}^{max}=10$, $n_p=10^6$, and $n=24$), we have $\delta_{atom}=\sqrt[480]{0.995}\approx 1-10^{-5}$. Then, according to Theorem 3, $n_e=47$ exchanges. This formula can be adapted to a churn-enabled context by modeling the churn and taking into account the evolving Newscast network topology.

B.1.2 Approximate Number of Noise-Shares

The noise-share generation algorithm is parameterized by n_ν, the expected total number of noise-shares that will be summed up. Ideally, this parameter shall be set to the number of participants involved in the sum. When the latter is predictable (*e.g.*, when there is no churn), n_ν can be set to the exact number of noise-shares that will be actually summed up. Otherwise, n_ν is not exact. In order to guarantee differential privacy, n_ν is set to the expected maximal lower bound on the number of noise-shares to be actually summed up. The actual sum thus involves more noise-shares than necessary. We show below that this surplus does not threaten the differential privacy guarantees.

LEMMA 3. *Let S denote the gossip approximate sum of values and $\widetilde{N}$ denote the gossip approximate sum of noise-shares.*

$\widetilde{N}$ *results from summing up more noise-shares than necessary, i.e.,* $\widetilde{N} = (1+e_{max}/(1-e_{max})) \cdot (1+e_N) \cdot (\sum_i^{n_\nu} \nu_i + \sum_i^{n_\nu^+} \nu_i)$ *where* n_ν *is the lower bound of noise-shares and* n_ν^+ *is the number of additional noise-shares. Then,* $S' = S + \widetilde{N}$ *satisfies* (ϵ, δ)*-probabilistic differential privacy, where* $\delta = (1-\iota)^2$.

PROOF. We have $\widetilde{N} = (1 + e_{max}/(1 - e_{max})) \cdot (1 + e_N) \cdot (\sum_i^{n_\nu} \nu_i + \sum_i^{n_\nu^+} \nu_i)$. Let a denote the noise-independent factors due to the gossip approximation, *i.e.,* $a = (1+e_{max}/(1-e_{max})) \cdot (1+e_N)$. Then $\widetilde{N} = a \cdot (\sum_i^{n_\nu} \nu_i + \sum_i^{n_\nu^+} \nu_i)$, and consequently $\widetilde{N} = a \cdot \mathcal{L}(\lambda) + a \cdot \sum_i^{n_\nu^+} \nu_i$. By adding $\widetilde{N}$ to S we have $S' = S + \widetilde{N} = S + \overline{N} + a \cdot \sum_i^{n_\nu^+} \nu_i$. Since (1) $S + \overline{N}$ satisfies (ϵ, δ)-probabilistic differential privacy where $\delta = (1-\iota)^2$ (Lemma 2), and (2) $a \cdot \sum_i^{n_\nu^+} \nu_i$ is independent from S and $\overline{N}$, then $S' = S + \widetilde{N}$ satisfies (ϵ, δ)-probabilistic differential privacy too. □

Additionally, participants can correct the sum of noise-shares $\widetilde{N}$ by subtracting a quantity equivalent in expectation to the non-necessary noise-shares included in the sum. Similarly to the additional noise-shares summed up in $\widetilde{N}$, the correction is independent from S and $\overline{N}$. Thus, its subtraction does not threaten differential privacy.

B.2 Security Against Inferences

We assess formally the security of Chiaroscuro against inferences according to the usual secure multi-party computation (SMC) methodology [14]. Informally speaking, it consists in comparing the information about the input dataset that leaks from the distributed algorithm instantiated on the distributed architecture (*real setting*), to the information that leaks from a centralized version of the algorithm run by an *ideally* trusted third party (*ideal setting*). If the difference between the two is negligible, then the distributed algorithm is said to be secure. The comparison is usually done based on the notion of *computational indistinguishability* [14], which states roughly that two distribution ensembles are computationally indistinguishable (denoted by $\stackrel{c}{\equiv}$) if no probabilistic polynomial-time algorithm is able to draw a significant difference between them.

We first formalize the security model of Chiaroscuro against inferences from honest-but-curious participants, which was informally stated in Section 2.3. Based on this model, we then give the full-length proof of security against inferences of Theorem 2 (stated in Section 4.3).

In the real setting, the adversary is (within) a participant. We follow the standard practical assumption that the adversary is computationally-bounded and model it as a probabilistic polynomial-time algorithm. The real-setting adversary has thus access to the participant's local data and parameters, to local transient information used or generated during the execution, and to the output of the algorithm. He exploits all this in any computationally-feasible way in order to gain additional knowledge about the other input time-series. In the ideal setting, the adversary has only access to the output of the centralized k-means algorithm perturbed by a differentially-private mechanism.

Loosely speaking, our security model (Definition 7) states that Chiaroscuro securely computes k-means if the side-effect information disclosed during the execution of an instance of Chiaroscuro (1) is perturbed by a differentially-private mechanism, (2) is independent from the input time-series and with the noise, or (3) does not increase significantly the adversarial knowledge (as defined by computational indistinguishability).

DEFINITION 7. *Let* π *be an instance of Chiaroscuro executed on a set of participants and* Δ *be the union of (1) the set of all perturbed means (either disclosed during* π *or output by the ideal-setting k-means) and (2) the set of all the information independent from the input time-series and noise. Given an input set of time-series* S*, an attacker* $\mathtt{A}$*, and an arbitrary background knowledge* $\chi \in \{0,1\}^*$*, we denote* $\mathtt{REAL}_{\pi,\mathtt{A}(\chi,\Delta)}(S)$ *the distribution representing the adversarial knowledge over the input dataset in the real setting - with a full knowledge of* Δ*, and* $\mathtt{IDEAL}_{kMeans,\mathtt{A}(\chi,\Delta)}(S)$ *the distribution representing the adversarial knowledge in the ideal setting - with a full knowledge of* Δ *too. We say that* π securely computes *k-means iff for every adversary* $\mathtt{A}_r$ *attacking* π*, there exists an adversary* $\mathtt{A}_i$ *for the ideal model so that for every* $\chi \in \{0,1\}^*$*:*

$$\{\mathtt{REAL}_{\pi,\mathtt{A}_r(\chi,\Delta)}(S)\}_S \stackrel{c}{\equiv} \{\mathtt{IDEAL}_{kMeans,\mathtt{A}_i(\chi,\Delta)}(S)\}_S$$

PROOF. The security of π against honest-but-curious participants can be shown through a thorough and exhaustive examination of the data structures that are communicated among participants along the execution sequence in the real setting. We focus on the computation step because the assignment step and the convergence step at each iteration are performed locally, by each participant, without any data transfer.

1. Epidemic computation of the encrypted means: the semantically-secure encryptions $\mathfrak{M}.\mathfrak{f}$ and $\mathfrak{M}.\mathfrak{c}$, and the data-independent $\mathfrak{M}.\omega$ and n;
2. Epidemic noise generation: *idem*, and the data-independent `ctr` and `cor`;
3. Epidemic decryption: the partially encrypted $\mathfrak{M}.\mathfrak{f}$ and $\mathfrak{M}.\mathfrak{c}$, the perturbed $\mathcal{M}$, and the data-independent *idk*;

We can easily observe that (1) any information depending on the personal time-series is communicated either after having been encrypted by a semantically-secure encryption scheme (step 1) or after having been perturbed by a differentially-private mechanism (step 3), and (2) *idem* for the information depending on the value of the noise generated (step 2). The remaining data structures depend neither on the time-series nor on the noise. In other words, by denoting by $\mathcal{E}$ the set of information encrypted by a semantically-secure encryption scheme (which output is computationally-indistinguishable from the output of a pseudo-random bitstring generator) : $\mathcal{E} = \{\mathfrak{M}.\mathfrak{f}, \mathfrak{M}.\mathfrak{c}\}$ and $\{\mathfrak{M}.\omega, n, \mathtt{ctr}, \mathtt{cor}, idk, \mathcal{M}\} \in \Delta$.

As a result :

$$\{\mathtt{REAL}_{\pi,\mathtt{A}_r(\chi,\Delta)}(S)\}_S \stackrel{c}{\equiv} \{\mathtt{IDEAL}_{kMeans,\mathtt{A}_i(\chi,\Delta,\mathcal{E})}(S)\}_S$$

Moreover, since the ideal-setting adversary can simulate $\mathcal{E}$ through a pseudo-random bistring generator, then :

$$\{\mathtt{IDEAL}_{kMeans,\mathtt{A}_i(\chi,\Delta,\mathcal{E})}(S)\}_S \stackrel{c}{\equiv} \{\mathtt{IDEAL}_{kMeans,\mathtt{A}_i(\chi,\Delta)}(S)\}_S$$

Consequently :

$$\{\mathtt{REAL}_{\pi,\mathtt{A}_r(\chi,\Delta)}(S)\}_S \stackrel{c}{\equiv} \{\mathtt{IDEAL}_{kMeans,\mathtt{A}_i(\chi,\Delta)}(S)\}_S$$

□

B.3 Collusions

The degree of resistance to collusions is defined by the number of key-shares and noise-shares used. Compromising a participant results essentially in disclosing its key-share and its noise-share. First, the disclosure of a key-share leaks nothing about the secret polynom in Ξ if the attacker has not $\tau - 1$ other key-shares. Second, the disclosure of a noise-share leaks the smallest possible fraction of noise (the number of noise-shares is maximal because noise-shares are distributed on the complete population). For example, with one million participants, and a participant colluding with $c - 1$ others, there remains $10^6 - c$ unknown other noise-shares used for generating the noise. As a result, the fraction of the secret noise in Ξ unknown to a colluding participant p decreases linearly with the number of collusions from p (bounded by the given threshold τ).

C. CORRECTNESS

Chiaroscuro follows an iterative execution sequence. Each iteration has three steps : the assignment step, the computation step, and the convergence step. We start by showing the correctness of both the assignment step and the convergence step. Then we concentrate on the correctness of the computation step.

C.1 Assignment Step, Convergence Step, and Termination

During the assignment step, each participant computes the distance between its time-series and each of the centroids output by the previous iteration (they appear in the clear, their privacy being guaranteed by differential privacy) to identify the closest centroid. During the convergence step, each participant computes the distance between the new set of centroids (resulting from the previous computation step) and the old ones (output by the previous iteration) and compares it to the given convergence threshold. These two steps (1) involve local computations only (2) based on cleartext data. They are consequently trivially correct.

The k-means iterations stop either when centroids have converged or when a fixed number of iterations has been reached. This disjunctive termination criterion is easy to check locally and guarantees that the k-means algorithm terminates.

C.2 Computation Step

C.2.1 Epidemic Computation of the Encrypted Means

First, the encrypted sum and count vectors of the means (*i.e.,* $\mathfrak{M}.s$ and $\mathfrak{M}.c$) are computed by the `EESum` algorithm, our gossip-based algorithm dedicated to computing homomorphically-encrypted sums. While the local update rule of the standard non-encrypted gossip-based sum algorithm [23] performs a division by two at each exchange, the local update rule of the `EESum` algorithm (Algorithm 2) simply delays it to the end of the algorithm (homomorphic encryptions do not support divisions). It is easy to see that both update rules are arithmetically equivalent.

PROOF. With the standard non-encrypted local update rule, a given variable will have been divided by 2^i after its i^{th} average. This is the same with our homomorphically-encrypted update rule. Indeed, we keep track (1) of the encrypted sum of variables and (2) of the divisor. First, by definition, the divisor is always equal to 2^{n_l}, where n_l is the current number of exchanges. Second, all variables having been averaged less than n_l times, say i times, are multiplied by 2^{n_l-i} during the exchange (*i.e.,* the scaling operation in the update rule). As a result, their actual divisor at exchange n_l is equal to $2^{n_l}/2^{n_l-i}=2^i$ where i is the current number of times they have been averaged. This is arithmetically equivalent to performing directly the average on values. The local update rule of the `EESum` algorithm is thus correct. □

The `EESum` algorithm terminates after a maximum number of exchanges fixed beforehand to a value sufficient for the sum to converge. Since the actual number of exchanges performed by each participant increases monotonically, the termination criterion is guaranteed to be satisfied eventually. Since it can be trivially enforced locally by each participant, the `EESum` algorithm is thus guaranteed to terminate.

C.2.2 Epidemic Noise Generation

The epidemic noise generation step computes the sum of noise-shares based on the `EESum` algorithm and corrects it if necessary by disseminating the correction vector associated to the lowest identifier. The `EESum` algorithm has been shown above to be correct; we focus on the correctness of the dissemination algorithm. First, each participant having contributed to the sum of noise-shares independently computes a correction proposal and assigns it a random identifier. Next, participants disseminate it based on standard epidemic dissemination algorithms, keeping at each exchange the correction proposal with the lowest identifier. Because the set of corrections (and identifiers) is fixed (only the participants involved in the sum can propose a correction vector), the standard diffusion speed results of epidemic dissemination algorithms apply [23]. Similarly to gossip aggregation algorithms, the probability that the disseminated correction vector does not reach a part of the population (1) depends on the number of gossip exchanges and (2) converges to zero exponentially fast (*e.g.*, in our experiments, a value was disseminated to one million participants with less than 50 messages per participant). We consider it as negligible in practice.

C.2.3 Epidemic Decryption

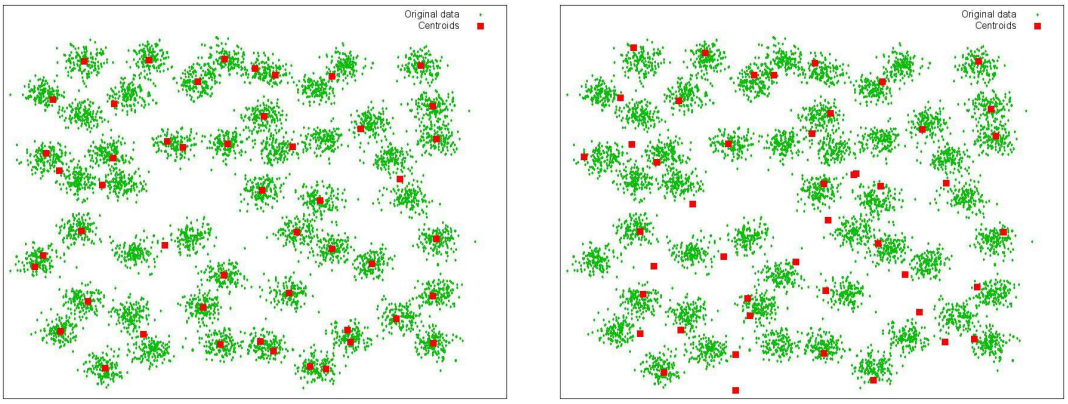
Finally, the correctness of the epidemic decryption is easy to state. First, a given key-share is applied at most once to a given encrypted value. Indeed, each participant keeps the identifiers of the key-shares having been applied to his local encrypted value. Second, at most n_κ key-shares are applied to each encrypted value. By counting the number of key-shares identifiers kept locally, any participant knows when n_κ is reached. Finally, the local sets of key-shares grow monotonically, guaranteeing the termination of the epidemic decryption. Indeed, by definition, no key-share is removed from a local set.

D. ILLUSTRATION ON 2D POINTS

In order to further illustrate the results of Chiaroscuro, we have used a $750K$ two-dimensional points dataset. These points are distributed into a set of 50 clusters[12]. We have executed both the standard k-means algorithm and Chiaroscuro (GREEDY strategy) over it with the parameters used in the evaluation section (Section 6). Our version of this dataset was generated by duplicating 100 times each of the $7.5K$ points contained in the A3 dataset and adding to each copy a uniform random value small enough to preserve the clusters. Two-dimensional points differ from time-series in the absence of temporal relationship between dimensions (though they are similar to time-series of size 2 for the privacy part). Therefore, the quality-enhancing smoothing techniques presented in Section 5.2 cannot apply.

Figures 6(a) and 6(b) represent the set of two-dimensional points (*i.e.*, small green points) (1) with the centroids obtained by the non-private standard k-means algorithm for the former figure, and (2) with the centroids obtained by the perturbed k-means algorithm (GREEDY strategy) for the latter figure. Both centroids were produced at the iteration #6, *i.e.*, the highest-quality iteration for the perturbed k-means. Although the perturbed centroids are less accurate, they appear mostly within an actual cluster, close to it, or between several clusters, which is similar to the usual behaviour of the non-perturbed version of k-means.

[12]I. Kärkkäinen and P. Fränti, "Dynamic local search algorithm for the clustering problem", Research Report A-2002-6, available at `https://cs.joensuu.fi/sipu/datasets/`



(a) In the Clear

(b) Chiaroscuro (GREEDY, no smoothing)

Figure 6: Centroids Resulting from k-means Executions over the $750K$ Two-Dimensional Points (6^{th} iteration)