**Noname manuscript No.**
(will be inserted by the editor)

# MET$_{\mathbb{A}}$P : Revisiting Privacy-Preserving Data Publishing using Secure Devices

**Tristan Allard · Benjamin Nguyen · Philippe Pucheral**

the date of receipt and acceptance should be inserted later

**Abstract** The goal of Privacy-Preserving Data Publishing (PPDP) is to generate a sanitized (i.e. harmless) view of sensitive personal data (e.g. a health survey), to be released to some agencies or simply the public. However, traditional PPDP practices all make the assumption that the process is run on a trusted central server. In this article, we argue that the trust assumption on the central server is far too strong. We propose MET$_{\mathbb{A}}$P, a generic fully distributed protocol, to execute various forms of PPDP algorithms on an asymmetric architecture composed of low power secure devices and a powerful but untrusted infrastructure. We show that this protocol is both correct and secure against *honest-but-curious* or *malicious* adversaries. Finally, we provide an experimental validation showing that this protocol can support PPDP processes scaling up to nation-wide surveys.

## 1 Introduction

### 1.1 Current PPDP Practices

In a traditional Privacy-Preserving Data Publishing (PPDP) process (Figure 1), *personal data* (e.g., patients' health data) is collected by a *publisher* who sanitizes it and releases it to various *data recipients* (e.g., research teams, public agencies, drug companies). Different sanitized releases are usually built by the publisher to best match (1) the data utility requirement expressed by each recipient and (2) the privacy requirement related both to the purpose of

T. Allard
Distributed Systems Laboratory, Univ. Politécnica de Madrid, Madrid, Spain
E-mail: tallard@fi.upm.es

B. Nguyen · P. Pucheral
SMIS Project, INRIA Rocquencourt, Le Chesnay, France
PRISM Laboratory, Univ. Versailles-St-Quentin, Versailles, France
E-mail: first.last@inria.fr

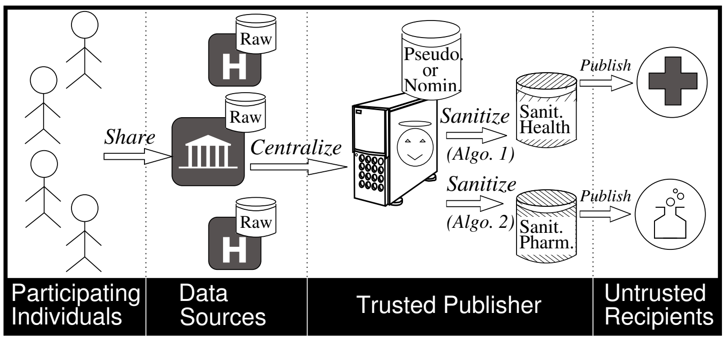


**Fig. 1** Current PPDP Practices

this release and to the trust level attached to this recipient. Although a lot of work has been done on privacy models trying to reach the best utility/privacy trade-off of the sanitized dataset (see [13,18] for two surveys), much less effort has been spent on the practical implementation and deployment of a PPDP process. This raises two major issues.

First, most PPDP works rely on the assumption of a trustworthy central publisher [13]. This trust assumption is unfortunately rarely satisfied in practice. There are many examples of privacy violations arising from negligence, abusive use, internal attacks and external attacks on servers[1]. Even the most secured servers are not spared [23], increasing individuals' fear of data centralization. When the data to be sanitized is already centralized (e.g., in a single-source Electronic Health Record system or EHR), the PPDP process does not worsen the situation. However, there are more and more situations where this data is scattered among several sources, either because data is of different nature and needs to be combined in the same release (e.g., social and medical data) or because data is hosted by autonomous sites (e.g., clinics, doctor's offices, regional hospitals) which are highly reluctant to externalize it in a single shared space. Therefore imposing a new centralization step for the sole PPDP purpose both hurts the course of history and is a new incentive to attack the publisher where high value aggregated data is henceforth stored.

Second, the legislation regulating the management of personal data [2], once sanitized, is rather permissive (no retention limit, no encryption of data at rest, no user consent), since it considers that this data is no longer sensitive. Some publishers exploit this flaw to implement highly practical - but somehow deviant - PPDP scenarios. For example, French and UK EHR providers, build sanitized releases as follows: (1) nominative data is extracted from an OLTP EHR server, (2) the data is simply pseudonymized (a legal form of sanitization!) and stored in a warehouse, and (3) different sanitized datasets are produced using this warehouse for different recipients on demand. We argue that letting the publisher host weakly sanitized data with weak security obli-

[1] `http://datalossdb.org/`

[2] For example, the European directive 95/46/EC.

gations is a major concern, at least as important as the privacy guarantees provided by the final release.

At a time where the scientific community mobilizes a lot of energy in designing privacy models to return to individuals a better control over their personal data, we advocate a fully decentralized PPDP process where individuals stay in the center of the loop (Figure 2). For the individual, this means (1) having the ability to opt-in/out of her participation to a given release according to its purpose and her confidence in the recipient and (2) keeping control over the process producing this release.

### 1.2 The Rise of Decentralized Alternatives

Fueled by the distrust of individuals towards central servers, distributed alternatives to a systematic centralization of personal data are rising in various domains. All these alternatives rely on the emergence of a growing family of client devices reaching high security standards thanks to secure hardware. In the medical domain, personal health care folders embedded in smart tokens[3] are receiving a growing interest (e.g., Health eCard[4] in UK, eGK card[5] in Germany, LifeMed card[6] in the USA, DMSP in France[7]) either to store primary copies of medical records or secondary copies of data hosted by EHR systems. More generally, the use of smart tokens is actively investigated by many countries for e-governance applications (citizen card, driving license, transportation, education, etc). The concept of a complete personal data server hosted in a secure device (either a plug computer or a smart token) and managing any kind of personal data under the holder's control has been popularized by the FreedomBox initiative[8] and the Personal Data Server approach [3]. Smart tokens have different form factors (e.g., SIM card, USB token, Secure MicroSD [20]) and names (e.g., Personal Portable Security Device [28], Smart USB Token [16], Portable Security Token [20] or Secure Portable Token [3]). Despite this diversity, smart tokens share similar characteristics (low cost, high portability, high storage capacity, high security), holding the promise of a real breakthrough in the management of personal data. But today smart tokens no longer hold a monopoly of client-side hardware security. The TrustZone technology pushed by ARM which integrates SecurCore tamper resistant elements is disrupting the design of secure systems by pushing hardware security to any mobile device (e.g., tablets, smartphones) equipped with Cortex-A processors[9]. According to ARM, a full Trusted Execution Environment (TEE)

[3] Roughly speaking, a smart token is the combination of a tamper resistant smart card micro-controller with a mass storage (gigabytes-sized) NAND Flash chip.

[4] `http://www.healthecard.co.uk`

[5] `http://www.gematik.de`

[6] `http://www.lifemedid.com/`

[7] `http://www-smis.inria.fr/DMSP/home.php`

[8] `http://www.freedomboxfoundation.org/`

[9] `http://tinyurl.com/ArmTrustzoneReport`

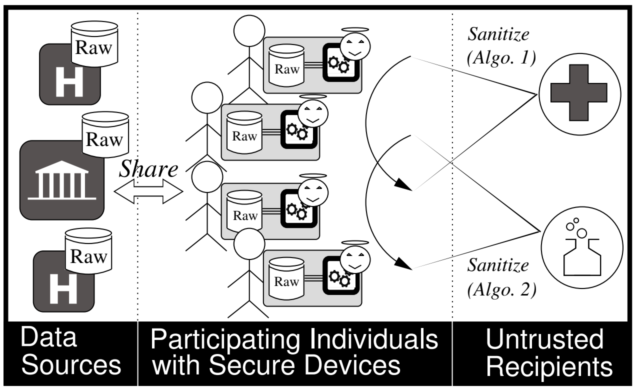


**Fig. 2** Proposed Approach

could soon be present in any client device at low cost. In this paper, and up to the experiment section, we consider that the individuals participating (henceforth called *participants*) are equipped with secure devices but make no additional assumption on the hardware solution they rely on.

## 1.3 Objectives

The challenge addressed in this paper is to devise a PPDP protocol compatible with the fully distributed setting provided by secure devices on the participant's side of the protocol. The objective is to get rid of the salient point of vulnerability introduced by a central publisher while still providing equivalent PPDP services. The implication is threefold:

- *Quality of the release:* implementing a distributed PPDP protocol[10] based on *local perturbation* [1] would be rather trivial (i.e., participants perturb their own data *independently*). However, a much better utility/privacy trade-off is reached by *global publishing* algorithms, also called *centralized publishing* algorithms [47], which exploit the prior knowledge of the complete dataset to be sanitized (e.g., data distribution) in order to perform a smarter sanitization. While fully decentralized, our protocol must preserve the quality reached by global publishing.
- *Genericity:* different sanitized releases must be delivered to different recipients depending on the utility/privacy trade-off required by each one. The proposed protocol must therefore be agnostic with respect to the targeted PPDP algorithms and their associated sanitization models (e.g., $k$-Anonymity, $\ell$-Diversity, Differential Privacy).
- *Scalability:* to be usable, the approach must scale up to nationwide sized datasets (millions of participants, each using a secure device).

[10] In order to avoid ambiguity we use the term *protocol* to denote the distributed implementation of an algorithm.

As Figure 2 suggests, the required distributed PPDP protocol looks like a Secure Multi-party Computation (SMC) protocol. Participants (i.e., secure devices) must jointly compute a function (i.e., the sanitization process) without revealing their input (i.e., the personal data they host), such that only the final result (i.e., the sanitized release) can be observed by the target recipient. However, no state of the art SMC protocol tackles our problem accurately. Generic SMC approaches (e.g., [51]), capable of computing arbitrary functions, have an exponential cost in the input size : they do not achieve the *scalability* objective. Conversely, specific PPDP SMC protocols (e.g., [54,31]) fail to answer the *genericity* objective; they additionally make very restrictive assumptions on the attack model and on the communication model which are incompatible with our own setting.

The solution promoted in this paper matches the three aforementioned requirements by exploiting the hardware security of the client devices, so that participants in the PPDP protocol can henceforth trust each other. The emergence of secure devices has already motivated the re-examination of a number of traditional SMC problems (e.g., [29,25,17]), pushing back well-established limits. To the best of our knowledge, the first study of a secure device-based approach to privacy-preserving data publishing is our own work [4,5]. However, the algorithm proposed in this preliminary work was not generic ($k$-Anonymity by generalization) and considered a central untrusted publisher. Hence this paper is the first proposal achieving a generic, fully distributed, and secure solution.

The paper is organized as follows. Section 2 introduces the hypothesis of our study and precisely states the problem to be solved. Section 3 discusses the related works by first presenting the targeted PPDP algorithms and the running examples, and second positioning the state of the art solutions with respect to the hypothesis of our study. Section 4 presents Met$_{\mathbb{A}}$P, our global approach to define a generic and scalable PPDP meta-protocol matching the characteristics of the targeted distributed architecture. Section 5 discusses the correctness of Met$_{\mathbb{A}}$P and Section 6 discusses its security against honest-but-curious and malicious adversaries. Finally, Section 7 demonstrates the practicability of the approach through experiments and Section 8 concludes.

## 2 Problem Statement

This section aims at defining precisely each element of the problem addressed. We first describe the distributed architecture for which Met$_{\mathbb{A}}$P is designed, characterizing the set of secure devices on the one side, the untrusted recipient on the other side, modeling the corresponding threats, and explaining why we discard the approaches based on server-side secure hardware. Second, we give the classes of algorithms targeted by Met$_{\mathbb{A}}$P and motivate the need for a wide scope (genericity requirement). Third, we detail the correctness and security models that Met$_{\mathbb{A}}$P must fulfill. We finally state the problem based on the previous definitions.

## 2.1 The Asymmetric Architecture

We introduce the Asymmetric Architecture, composed of secure devices (Section 2.1.1) and Recipients (Section 2.1.2). We then present three possible threat models, of increasing adversarial power (Section 2.1.3), and conclude with a reflection on the inadequacy of secure server side solutions with regards to these threat models (Section 2.1.4).

### *2.1.1 The Secure Devices*

As stated in the introduction, we assume that each participant of the PPDP protocol, is equipped with a secure device hosting a (primary or secondary) copy of her personal data. As pictured in Figure 3, and despite the diversity of existing hardware solutions, a *secure device* can be abstracted by (1) a Trusted Execution Environment and (2) a (potentially untrusted) mass storage area. For example, the former can be provided by a tamper-resistant micro-controller while the later can be provided by a SD card. The important assumption is that the code executed by the secure device cannot be tampered. This given, the content of the mass storage area can be easily protected using cryptographic protocols. Each secure device thus exhibits the following properties:

**High Security Guarantees.** Compared to a traditional server, the very high security provided by secure devices comes from a combination of factors: (1) the tamper-resistance of its processing unit making hardware and side-channel attacks highly difficult, (2) the certification of the hardware and software platform[11], making software attacks (e.g., trojan) also highly difficult, (3) the capacity to be auto-administered, contrary to high-end multi-user servers, avoiding insider (i.e., DBA) attacks, (4) the obligation to physically be in contact with the device to attack it, and (5) even the device's owner cannot directly access the data stored locally nor spy the local computing (she must authenticate, using a PIN code or a certificate, and only gets data according to her privileges).

**No Availability Guarantee.** A secure device provides *no guarantee of availability* except to its owner who physically controls it, connecting and disconnecting it at will.

**Modest Computing Resource.** Most secure devices provide *modest computing resources* (see Section 7) due to the hardware constraints linked to their tamper-resistance. On the other hand, a dedicated cryptographic coprocessor usually handles symmetric cryptographic operations very efficiently (e.g., `AES` and `SHA`). Hence, even if differences exist between secure devices in terms of tamper-resistance (e.g., smart tokens are more robust against tampering than TrustZone enabled devices) and computing power (e.g., smart tokens are much less powerful than TrustZone devices), all provide much better security guarantees combined with much lower availability and computing power than a traditional server.

[11] `http://www.commoncriteriaportal.org/`

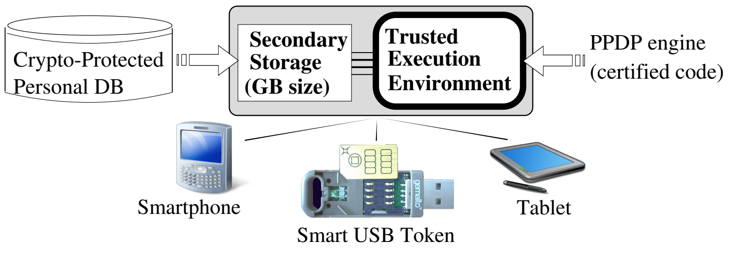

Crypto-Protected Personal DB
Secondary Storage (GB size)
Trusted Execution Environment
PPDP engine (certified code)
Smartphone
Smart USB Token
Tablet

**Fig. 3** Secure Devices

Finally, although hardware protected servers exist, we discuss in Section 2.1.4 why we do not consider they solve the problem.

### *2.1.2 The Recipients*

Due to their weak availability and modest computing resources, Secure Devices cannot implement a global publishing algorithm on their own (i.e., in a peer-to-peer fashion as in most SMC protocols). First, they require external communication facilities (i.e., a mailbox-like service) to handle the asynchrony of their connections and allow inter-device communications. Hence, sanitization of large datasets can be split into small computation units, each taken in charge by autonomous devices, and intermediate results are exchanged via this communication service. Second, global publishing algorithms reach high utility/privacy ratio by exploiting the knowledge of the complete dataset to be sanitized. Depending on the PPDP algorithm, gaining and exploiting this knowledge over a very large dataset in a fully distributed way with low power devices may be unrealistic (e.g., this may require to sample or classify the complete dataset). To tackle this issue, we consider delegating a controlled amount of computation to an external computing service, in such a way that this computation cannot breach the privacy of the participants.

We do the assumption that the required communication and computing services are provided by the recipients themselves, through traditional servers[12]. The benefit of this assumption is that it avoids reintroducing a single central entity in the architecture. Each recipient exhibits the following properties:

**Low Security Guarantees.** As any traditional server, a recipient is prone to privacy violations due to negligence, internal and external attacks.

**High Availability Guarantee.** A recipient relies on traditional server capabilities : it is assumed to provide 24/7 availability.

**High Computing Resource.** A recipient relies on traditional server capabilities and therefore does not suffer any computing restriction.

Since this architecture is composed on the one hand of a very large number of low power, weakly connected but highly trusted devices and on the other

[12] For example, a recipient (e.g., a research team) may either use its own server, the computing facilities of a larger institution (e.g., an hospital) or even externalize these services to a third party (e.g. a Cloud provider) assuming this third party formally commits to providing the same privacy guarantees as the ones imposed to the recipient itself.

hand of powerful external computing and communication resources provided by untrusted recipients, we call it an *Asymmetric architecture.*

### *2.1.3 Threat Model*

In an ASYMMETRIC architecture, the main source of vulnerability comes from the recipients (they can be dishonest or be themselves attacked). Secure devices are presumably trusted. However, even secure hardware can be breached, though at very high cost. Therefore we take into account the possibility that a very small number of participating secure devices might be compromised. We thus consider three attack models of increasing strength.

– *Honest-but-curious*: the attacker does not deviate from the protocol it is participating in but tries to infer confidential data (in principle hidden by the protocol) by exploiting in any computationally feasible way the results of each step of the protocol. In this model, the attacker (also called *semi-honest* or *passive*) is the recipient.
– *Malicious*: in this model, the attacker is still the recipient, therefore we do not consider denial of service attacks : the attacker will cheat the protocol with the sole objective to disclose confidential data. This model assumes that the recipient can now conduct passive *and* active attacks (i.e., modify the results of some steps of the protocol). Avoiding active attacks is impossible since the recipient plays a role in the protocol but being able to detect active attacks is considered as the best way to deter them[13]. Hence, the recipient can be said a malicious adversary having weakly-malicious intent [52] : it will perform active attacks only if (1) it will not be convicted as adversary by the trusted parties and (2) the final result is correct. Note that the secure devices form the trusted parties.
– $Malicious_{Hard}$: the attacker is *malicious* and is able to break the tamper-resistance of *at least one* Secure Device and disclose its internal cryptographic material. In this model, the attacker is the recipient colluding with the compromised device(s).

For clarity, these models will be considered one after the other throughout this article, with a particular focus on honest-but-curious and malicious attacks. The way $Malicious_{Hard}$ attacks are tackled will be sketched as an implementation variation of our protocol.

### *2.1.4 Inadequacy of Server Side Secure Hardware*

Since secure hardware is considered on the participant's side, why not consider secure hardware also on the recipient side? Powerful (and costly) secure hardware exists and could act as a trusted proxy for the server (here, the recipient)

[13] The detection of an attack puts the attacker in an awkward position. If the data leak is revealed in a public place, participants are likely to refuse to participate in further studies with an irreversible political/financial damage and they can even engage a class action.

[7]. Such an assumption would greatly simplify the protocol: all participants could send their raw data to this trusted proxy, which would implement the complete PPDP protocol and deliver the final result to the recipient. However, this would reintroduce a centralized trusted party in the architecture, and thus a clear incentive to attack it. For the same reason, we rule out any solution based on a very small number of secure devices (e.g., powerful secure clients running the PPDP protocol). The solution proposed in this paper adopts the opposite approach: it primarily relies on a wide distribution of data storage and processing on low cost personal secure devices and tries to get rid of the single point of vulnerability introduced by any central entity.

## 2.2 Targeted PPDP Algorithms

There are two major classes of models tackling the global (centralized) publishing problem, namely partition-based (e.g., $k$-Anonymity, $l$-Diversity ) and differential privacy (e.g., $(d, \gamma)$-Privacy), with several models in each class and a profusion of algorithms implementing each model.

Because of the differences exhibited by the datasets, the adversaries and the utility expected by the recipients, no single model/algorithm couple can legitimately claim to be *the golden standard* for data privacy. In recent works, Kifer *et al.* clarify the adversarial and data assumptions usually made by existing models and algorithms [33,35]. These works conclude that (1) *no* existing model/algorithm protects against any type and amount of adversarial background knowledge [33,35,14], and (2) overall, a model free from any data and adversarial assumption would be unable to output any useful PPDP release [35]. Motivated by this call for a diversity of models, the emerging unified theory of privacy and utility aims at providing formal guidelines for the design of rigorous models depending on such assumptions [34,36]. The universal PPDP model is thus a myth, as is the universal PPDP algorithm. Hence, the diversity of privacy models and data usages make paramount the support of a variety of privacy algorithms using an Asymmetric architecture. Moreover, the targeted objective is to provide a generic framework and methodology to help adapt any (existing or future) global publishing algorithm to the Asymmetric architecture.

## 2.3 Correct and Secure Computation

We call Met$_{\mathbb{A}}$P the implementation of our PPDP framework on an Asymmetric architecture. Met$_{\mathbb{A}}$P is actually a skeleton which can be instantiated with various PPDP algorithms. If we show that Met$_{\mathbb{A}}$P is correct and secure, then any instantiation of Met$_{\mathbb{A}}$P would inherit these same properties. Unfortunately, part of this proof depends on the information disclosed to the recipient by each algorithm's instantiation. Our methodology is therefore to prove once and for all the correctness and security of the generic part of Met$_{\mathbb{A}}$P, so that the attention can focus on the specific part of each instantiation. To this end, we follow the same strategy as for assessing the correctness and security of SMC protocols. Informally speaking, this strategy consists of showing that the SMC implementation of a given protocol and the *ideal* implementation of this same protocol are equivalent in terms of output and information leakage. We call *ideal* an implementation of a protocol by a centralized trusted third party (i.e., where dysfunctions linked to distribution and all forms of attacks are precluded).

**Correctness.** We say that the instantiation of a global publishing algorithm using $\text{Met}_{\mathbb{A}}\text{P}$ is *correct* if, whatever the input dataset, the distribution ensembles of its *outputs* in our setting (i.e. Asymmetric architecture) are computationally indistinguishable [21] from the distribution ensembles of the original algorithm's outputs in an ideal setting (i.e. trusted third party). In other words, the two results are equivalent (though they may not be equal strictly speaking due to the non determinism of PPDP algorithms).

**Security.** We say that the instantiation of a global publishing algorithm using $\text{Met}_{\mathbb{A}}\text{P}$ is *secure* if, whatever the input dataset, the amount of information disclosed by the execution of this instantiation does not increase the attacker's knowledge. In other words, the attacker learns nothing beyond its prior knowledge and the final release.

### 2.4 Problem Statement

In this article we tackle the problem of designing a decentralized PPDP protocol on an Asymmetric architecture that is:

- **Correct and secure:** its execution must produce an equivalent result with the same security guarantees as in an ideal setting, despite *honest-but-curious*, *Malicious* and $\textit{Malicious}_{Hard}$ attacks.
- **Generic**: the protocol must be agnostic with respect to the targeted PPDP algorithms and underlying privacy models.
- **Scalable**: the protocol must scale up to nationwide sized datasets.

We call $\text{Met}_{\mathbb{A}}\text{P}$ the resulting protocol, since it is a *meta-protocol* that needs to be instantiated on a specific PPDP algorithm $\mathbb{A}$.

Solving this problem is of paramount importance since this would lead, for the first time, to a realistic alternative to the increasingly disparaged centralized implementations of PPDP processes by so-called trusted publishers.

## 3 Related Works

This section presents the background knowledge needed to understand the paper. First, we introduce the two major PPDP approaches targeted by $\text{Met}_{\mathbb{A}}\text{P}$ and thoroughly describe the models and algorithms selected from both approaches and used along the paper as illustrative running examples. Then we position MET$_{\mathbb{A}}$P with respect to the state-of-the-art works: we overview the secure and distributed implementations of PPDP algorithms as well as the cryptographic protocols based on secure devices.

## 3.1 Targeted PPDP Algorithms

We start by presenting an overview of the major approaches to global publishing, then present three example algorithms illustrating the variety of PPDP models, $\alpha\beta$-ALGORITHM in Section 3.1.2, MONDRIAN in Section 3.1.3 and BUCKETIZATION in Section 3.1.4.

### *3.1.1 Overview of the Major Approaches to Global Publishing*

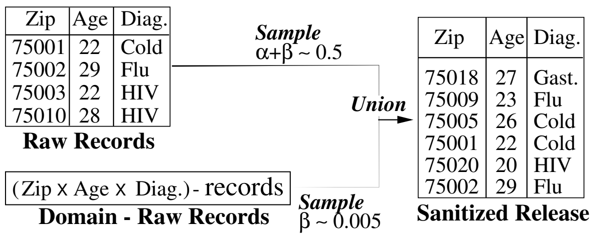
There are two major approaches to the global (centralized) publishing problem: the more recent *differential privacy approach* and the more traditional *partition-based approach.*

*The Differential Privacy Approach* The *differential privacy* approach, initiated in the interactive query answering setting [15], is receiving an increasing attention in global publishing (e.g., [47,42]). Loosely speaking, a differentially private algorithm is a *randomized* function which is defined such that the presence or absence of each participant's record in the initial dataset has only a quantifiably insignificant impact on the algorithm outputs. The main advantage set forward with the differential approach is that it gives quantifiable and provable guarantees on the information gained when viewing the release. In this article, we consider more specifically the $(d,\gamma)$-PRIVACY model [47] which is part of the differential privacy approach[14]. $(d,\gamma)$-PRIVACY models the adversarial knowledge as a probability distribution over the possible initial records and guarantees that after accessing the sanitized release, the adversarial knowledge about any record remains below a fixed bound. Note that the $(d,\gamma)$-PRIVACY model assumes that each record is unique.

We describe in Section 3.1.2 the $\alpha\beta$-ALGORITHM, a centralized publishing algorithm proposed in [47] for enforcing the $(d,\gamma)$-PRIVACY model.

*The Partition-Based Approach* Roughly speaking, the *partition-based* approach splits the attributes of the dataset in two categories: the *quasi-identifier* part and the *sensitive* part. A quasi-identifier (denoted $QID$) is a set of attributes for which some records may exhibit a combination of unique values in the dataset, and consequently be identifying for the corresponding participants (e.g., {ZipCode, BirthDate, Gender}). The sensitive part (denoted $SD$) encompasses the attribute(s) whose association with participants must be made ambiguous (e.g., {Disease}). Several quasi-identifiers, as well as several sensitive attributes, may exist in a dataset; but for simplicity, most of the literature

[14] In [46], the $(d,\gamma)$-PRIVACY model is shown to be equivalent to the $\epsilon$-INDISTINGUISHABILITY model [15].



| Zip | Age | Diag. |
|---|---|---|
| 75001 | 22 | Cold |
| 75002 | 29 | Flu |
| 75003 | 22 | HIV |
| 75010 | 28 | HIV |

| Zip | Age | Diag. |
|---|---|---|
| 75018 | 27 | Gast. |
| 75009 | 23 | Flu |
| 75005 | 26 | Cold |
| 75001 | 22 | Cold |
| 75020 | 20 | HIV |
| 75002 | 29 | Flu |

**Fig. 4** Illustration of the $\alpha\beta$-Algorithm

assumes the presence of a single quasi-identifier (possibly encompassing all of them) and of a single sensitive attribute.

Partition-based approaches essentially apply a controlled degradation to the association between participants (represented in the dataset by their quasi-identifier(s)) and their sensitive attribute(s). The initial dataset is deterministically partitioned into groups of records, (abusively) called *equivalence classes* in the rest of the paper, where quasi-identifier and sensitive values satisfy a chosen partition-based privacy model. The seminal $k$-Anonymity model [48] requires each class to contain at least $k$ records indistinguishable with respect to their (possibly coarsened) quasi-identifier values so that each sensitive data be associated with at least $k$ records. Successors of $k$-Anonymity (e.g., $\ell$-Diversity [40], $t$-Closeness [38], $\epsilon$-Privacy [39]) further constrain the distribution of sensitive data within each class, tackling different adversarial assumptions. The $\ell$-Diversity principle [40] requires that the set of sensitive data associated to each equivalence class be *diverse* enough, where the notion of diversity can be specified in various ways (the rationale being that the more diverse are the sensitive data, the less probable the association between a quasi-identifier and a given sensitive data of the class). In this article, we use the Mondrian algorithm (Section 3.1.3) and the Bucketization algorithm (Section 3.1.4) to illustrate the enforcement of, respectively, the $k$-Anonymity and the $\ell$-Diversity models.

### *3.1.2 The $\alpha\beta$-Algorithm Running Example*

The $\alpha\beta$-Algorithm (illustrated in Figure 4) enforces the $(d, \gamma)$-Privacy model in two simple steps: (1) sample the initial set of records with a sampling rate of $(\alpha + \beta)$ (we call the resulting records the *true* records) and (2) sample without replacement the definition domain of records, denoted *dom*, minus the initial dataset, with a sampling rate of $\beta$ (we call the resulting records the *fake* records). The algorithm finally publishes the domain, the parameters $\alpha$ and $\beta$, and the sanitized release. The privacy guarantees of the $\alpha\beta$-Algorithm arise from the mix of true and fake records in the sanitized release.

The $\alpha\beta$-Algorithm requires the knowledge of the complete dataset because of the *duplicate-free generation of fake records* that it performs: no fake record can be the duplicate of any other record, be it true or fake.

### *3.1.3 The Mondrian Running Example*

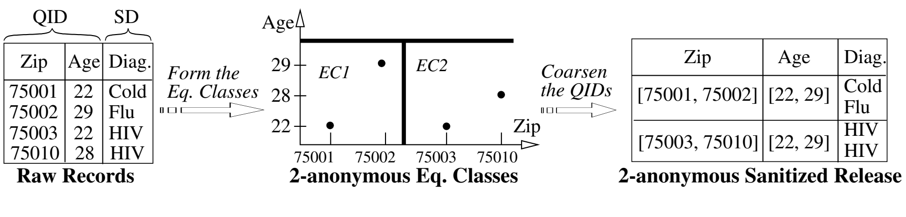
The MONDRIAN algorithm [37] is a well-known and highly intuitive partition-based algorithm originally proposed for enforcing $k$-ANONYMITY [48].

Conceptually, MONDRIAN (1) forms equivalence classes so that each class contains at least $k$ records whose quasi-identifiers are close and (2) coarsens the quasi-identifier of each record so that it covers (exactly) the other records of its class. MONDRIAN operates through a recursive splitting process. It starts with a single class that contains the complete dataset (i.e., the corresponding coarsened quasi-identifier would cover all the records) and splits it around the median of a selected dimension of the quasi-identifier. This yields two classes, each containing about half the records of the parent class: the corresponding coarsened quasi-identifiers are thus more accurate because each one covers less tuples. The recursive splitting process continues until no class can be further divided without containing less than $k$ records. The algorithm finally coarsens each record's quasi-identifier according to its class, and publishes the resulting (transformed) dataset. Figure 5 illustrates the MONDRIAN algorithm on a simple dataset.

The need to know the complete dataset prior to executing MONDRIAN is obvious from the above description: the recursive splitting process forms the equivalence classes based on the complete set of records.

### *3.1.4 The Bucketization Running Example*

The BUCKETIZATION algorithm proposed in [49] and depicted in Figure 6 forms equivalence classes so that each class (1) is $\ell$-diverse and (2) is split in two separate tables where one table holds the quasi-identifiers of the records in the class and the other one holds their corresponding ($\ell$-diverse) sensitive data. Indeed, unlike MONDRIAN that coarsens the quasi-identifiers, the association between quasi-identifier and sensitive data is degraded here through the lossy join between the two tables, preserving the inner distribution of quasi-identifiers in each class. The original BUCKETIZATION algorithm adopts a popular definition of $\ell$-DIVERSITY that states that within each class, the frequency of a sensitive value must not exceed $1/\ell$. The BUCKETIZATION algorithm simply operates by (1) grouping records by sensitive data, (2) recursively forming equivalence classes by picking randomly one record in each of the $\ell$



| Zip | Age | Diag. |
|---|---|---|
| 75001 | 22 | Cold |
| 75002 | 29 | Flu |
| 75003 | 22 | HIV |
| 75010 | 28 | HIV |

| Zip | Age | Diag. |
|---|---|---|
| [75001, 75002] | [22, 29] | Cold Flu |
| [75003, 75010] | [22, 29] | HIV HIV |

**Fig. 5** Illustration of the MONDRIAN algorithm

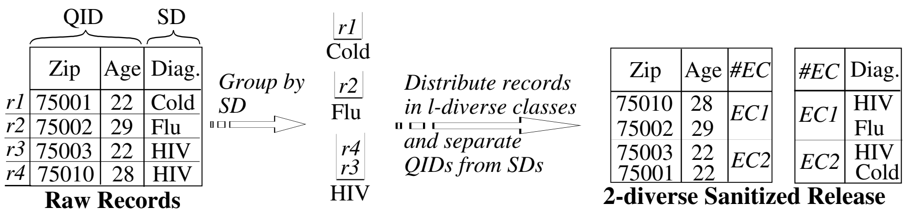


**Fig. 6** Illustration of the BUCKETIZATION algorithm

biggest groups (until less than $\ell$ groups remain), and finally (3) assigning each record of the remaining groups to a class that does not already contain its sensitive data.

The knowledge of the complete dataset is mandatory for the BUCKETIZATION algorithm because the appearance of each record in a given equivalence class depends on the sensitive data of the other records.

## 3.2 State-of-the-Art

To the best of our knowledge, no work has considered a context similar to this paper. The traditional *Secure Multi-party Computation* (*SMC* for short) approach neither considers secure devices nor is generic and scalable *simultaneously*. The only approach based on secure devices and tackling a PPDP context is our own previous work which is neither generic nor fully distributed. We explain these claims below.

### *3.2.1 Traditional Secure Multi-party Computation*

*Generic Secure Multi-party Computation* Early works such as, e.g., [51,22], have investigated methods for translating *any* centralized function to a decentralized one that provides SMC security guarantees. The resulting generic translation method essentially consists in expressing the centralized function as a combinatorial circuit and distributing its execution on the set of participants. The resulting cost depends on the number of inter-participant interactions (whose execution is mainly based on oblivious transfer protocols), which in turn depends exponentially on the size of the input data, on the complexity of the initial function, and on the number of participants. Despite their unquestionable theoretical interest, these generic approaches cannot be used in practical global publishing scenarios where inputs are large and sanitization algorithms complex.

*Secure Multi-party Computation for Global Publishing Algorithms* The non-practicality of the generic SMC approach has led to the investigation of SMC solutions specific to each problem. To the best of our knowledge, only a small number of works has focused on the global publishing problem [54,30,44,53,31,43,50], and in a way that severely limits their scope. First, they are not generic: their internals consist in cryptographic protocols specifically designed to enforce a given privacy model/algorithm ($k$-Anonymity or variants for most). Second, they do not fit the Asymmetric architecture. They make strong assumptions of availability or computing resources for participants, or reintroduce a central point of attack.

The authors of [54] aim at producing for a miner and without any central trusted server a $k$-anonymous release of the union of data held by a set of participants. The participants and the miner are honest-but-curious. Zhong *et al.* design a first solution letting the miner decrypt only the records whose raw quasi-identifiers appear more than $k$ times in the dataset (other records are lost). To avoid the possibly high data loss, a second solution lets the miner execute the partition-based algorithm proposed in [41] over encrypted records. To this end, participants disclose to the miner a controlled amount of information consisting of the following: for each pair of quasi-identifiers the number of attribute values that differ. Both solutions do not answer the objectives of this paper in that they are strongly tied to the $k$-Anonymity model (no genericity), they consider an honest-but-curious miner (no malicious attacker), and they are designed for fully connected participants (unfit to the Asymmetric architecture).

The approach proposed in [50] is similar to the second solution proposed in [54]: the difference mainly lies in the partition-based algorithm used to produce equivalence classes, i.e., [19], which is adapted by disclosing to the miner the relative orders and distances between quasi-identifiers. The proposed protocol between the participants and the miner is based on calls to *homomorphic* and *private information retrieval* schemes. Similar shortcomings preclude this approach.

In [53], Zhong *et al.* relax the high availability requirement made by [54] on the complete set of participants and consider a malicious miner. However, the protocol proposed is still unable to meet our objectives. First a single point of failure (attack) is reintroduced by requiring that a specific participant, called the *helper*, be fully connected and never collude with the miner. The complete protocol is a two-party protocol between the helper (that has collected the participants' data in an encrypted form) and the miner. Second, the internals of the proposed protocols are designed for the $k$-Anonymity model: participants' data is encrypted such that the two-party protocol lets the miner suppress the quasi-identifiers appearing less than $k$ times.

In [31], the authors propose an adaptation of the Mondrian algorithm to produce a $k$-anonymous release of the union of the datasets. However, the proposed protocol first does not fit the Asymmetric architecture (full availability requirement of the participants), second is strongly tied to Mondrian (it is a distributed adaptation of Mondrian's recursive splitting procedure), and third limits the attack model to honest-but-curious.

The approach proposed in [43] differs from [31] only in the underlying privacy model (a home-made variant of $k$-Anonymity) and algorithm (designed for enforcing the variant they propose). Both present similar shortcomings with respect to our context: hardcoding of the privacy algorithm into the internals of the protocol, a full availability requirement for the participants, honest-but-curious attack model.

Finally, the approaches proposed in [30] and [44] consider a context where data is vertically partitionned accross the set of participants. The resulting protocols consist in an iterative sanitization of the global dataset (consisting in the (virtual) join of the vertical partitions) through local proposals emitted by participants (each participant has the full set of records (projected on a subset of attributes)). In addition to the usual shortcomings (e.g., honest-but-curious attack model, assumption of few participants, full availability), the approaches do not fit a context where data is horizontally partitioned (each participant has only a subset of records and is consequently unable to reach local sanitization decisions). We note that [30] is able to translate usual centralized $k$-Anonymity algorithms into secure distributed two-party protocols: though the need to provide genericity was felt, genericity was limited here it to the $k$-Anonymity model and algorithms.

### *3.2.2 Approaches Based on Secure Devices*

The interest in cryptographic protocols founded on secure devices is resurfacing. Indeed, many works revisit the traditional approaches and their results, based on the use of secure hardware. For example, [32,24] circumvent the theoretical impossibility results of secure multi-party computation protocols by benefiting from the tangible trust provided by tokens; [29] benefit from tamper-resistant devices to decrease the communication cost of the generic SMC approach by equipping parties with such devices which are - locally - in charge of a part of the circuit evaluation; [27,17] propose two-party secure set intersection protocols based on a physical exchange of (preloaded) secure devices. To the best of our knowledge, the only approach to centralized publishing over an architecture based on secure devices is our own previous work [4,5]. However, this work (1) is limited to both the $k$-Anonymity model and the global recoding algorithms and (2) is not fully distributed because it uses a central entity as an intermediate between secure devices and the recipient. By reaching full genericity and getting rid of any intermediate central entity, the current paper addresses a problem unaddressed previously.

## 4 A Unified View of Global Publishing Algorithms

We overcome the shortcomings of the state-of-the-art SMC solutions by devising Met$_{\mathbb{A}}$P, an abstract PPDP protocol answering the problem statement presented in Section 2.4. The genericity of the protocol is achieved by defining a common execution sequence embracing the behaviour of traditional global publishing algorithms. Scalability is reached by subsequently adapting this sequence to an Asymmetric architecture, taking advantage of its intrinsic parallel processing capabilities. Then, the correctness and security properties of MET$_{\mathbb{A}}$P are proved, as a main step to guarantee the correctness and security of all concrete publishing algorithms instantiated through MET$_{\mathbb{A}}$P. Our approach is sketched below.

*Genericity* All global publishing algorithms we are aware of can be arranged in a succession of three phases as follows. The *collection phase* gathers the input dataset to be sanitized. In this phase, the publisher collects the information generated by each agreeing participant, until reaching the expected cardinality. Then, the *construction phase* analyzes the complete dataset and, according to this, produces the *sanitization information* which will maximize the utility/privacy tradeoff in the final release. Finally, the *sanitization phase* produces this final release, sanitizing each collected record individually according to this sanitization information. Putting aside the performance issue (which may introduce significant difference between PPDP algorithms), organizing the processing in such a way gives prominence to the fact that the specific part of each PPDP algorithm sums up to the computation of the sanitization information.

*Scalability* As mentioned before, the objective is to adapt these three phases to an ASYMMETRIC architecture, exploiting its parallel computing potential without sacrificing genericity. The collection phase is intrinsically parallel, each participating secure device sending its owner's record to the recipient with no synchrony requirement. The sanitization phase also can be easily computed with independent parallelism since it operates at a record granularity[15]. To make this possible, during the construction phase and once the sanitization information has been computed, the recipient partitions the dataset such that(1) each partition is a self-contained unit of treatment in that it contains both a subset of records and the sanitization information required to sanitize them and (2) the partition size fits the secure devices limited resources.The construction phase itself is unfortunately much more difficult to parallelize because (1) it executes at the dataset granularity and (2) it depends on the instantiated PPDP algorithm. As discussed in Section 2, no SMC implementation of such process could be envisioned without hurting either the genericity or scalability objective and could tackle the ASYMMETRIC hypothesis (asynchronous communications, passive and active attacks). To circumvent this difficulty, we delegate the computation of the construction process to the recipient which benefits from high computing resources and explain next how to make this correct and secure.

*Correctness and Security* MET$_{\mathbb{A}}$P must be shown correct (i.e., give a result equivalent to the one obtained in an ideal setting) and secure against *honest-*

[15] The degree of parallelism is determined by the number of secure devices which connect together during the sanitization phase, every token being eligible to participate to this phase, including those which did not participate to the collection phase. In the extreme case where secure devices connect one after the other, the processing will simply be performed sequentially.

*but-curious* and *malicious* adversaries. Tackling this requirement is not a trivial task. It is worth noting that this problem is not specific to Met$_{\mathbb{A}}$P but to any attempt to adapt a global publishing algorithm to an untrusted distributed context. Correctness and security aspects exacerbate the benefit of a generic protocol skeleton. Indeed, each *ad-hoc* adaptation of an algorithm would require re-starting the correctness and security analysis from scratch, identifying specific attacks, designing home-made counter-measures, and proving their security. Met$_{\mathbb{A}}$P only needs to be proved correct and secure once and for all. Whatever the instantiated algorithm, it will inherit all the correctness and security guarantees provided by Met$_{\mathbb{A}}$P for its generic part.

The proof of correctness relies on the demonstration that the result of each phase (collection, construction, sanitization) is similar to its centralized counterpart. Regarding security, both passive and active attacks must be considered at each phase. To prevent passive attacks, each record is initially encrypted by its hosting secure device - with a randomized encryption scheme - before being delivered to the recipient during the collection phase. Records remain encrypted until the very last step of Met$_{\mathbb{A}}$P. During this same collection phase, and to allow execution of the construction phase, an extra amount of information must be disclosed (i.e., sent un-ciphered) to the recipient by each secure device[16]. This information, called hereafter *construction information*, is algorithm dependent and must not increase the attacker's knowledge. Based on it, the recipient then analyzes the dataset, appends to each encrypted record its corresponding sanitization information, and partitions them. During the sanitization phase, secure devices finally download partitions one by one, decrypt their content, sanitize each record accurately and produce part of the final release. To simplify the presentation, we first assume that cryptographic keys are shared by all secure devices, so that any secure device can contribute equally to the sanitization phase (including those which did not participate in the collection phase). We relax this assumption in Section 6.3. Regarding active attacks, clearly defined *safety properties* have to be asserted along the protocol by the participating secure devices in order to detect any attempt of attacking the data (i.e., the collected dataset and the computed sanitization information) or the logic (i.e., the execution sequence). We sketch possible implementations for safety properties, emphasizing their feasibility rather than their optimality. Correctness and security issues are deeply discussed in Section 5 and Section 6 respectively.

*Methodology* Proving the correctness and security of Met$_{\mathbb{A}}$P is an important result of this paper, stated by Theorem 1, Theorem 2, and Theorem 3. Considering this given, instantiating Met$_{\mathbb{A}}$P using any global publishing algorithm and proving the correctness and security of the result leads to the following steps:

[16] A similar tradeoff occurs in the query outsourcing approach where an untrusted host must be able to issue queries over encrypted data (e.g., [26]).

1. to characterize which *construction information* must be mandatorily disclosed to the recipient to allow it in turn to compute the *sanitization information*;
2. to prove that this *contruction information* cannot be source of passive attacks;
3. to characterize the test that must be performed by the secure devices in order to assess the legitimacy of the *sanitization information* produced by the recipient. This legitimacy is linked to the dataset and to the privacy guarantees enforced by the instantiated algorithm;

The next section will illustrate this methodology by instantiating MET$_{\mathbb{A}}$P using our three running examples, namely the $\alpha\beta$-ALGORITHM, the BUCKETIZATION algorithm and the MONDRIAN algorithm.

## 5 A Correct Meta-Protocol

This section focuses on the correctness of the MET$_{\mathbb{A}}$P execution sequence, postponing to the next section its protection against honest-but-curious and malicious adversaries.

This section is organized as follows. First, we specify the format of (1) the tuples that, flowing from secure devices to the recipient, form the collected dataset and feed the construction phase, called *c-tuples*, (2) the tuples that, flowing from the recipient to secure devices, are to be sanitized, *called s-tuples*, and (3) the resulting sanitized tuples that are returned by secure devices, called *r-tuples.* Second, we detail each of the collection, construction, and sanitization phases. Third, we illustrate the genericity of the approach through a sample of supported PPDP algorithms. Finally, we formally show MET$_{\mathbb{A}}$P's correctness.

### 5.1 Basic Tuple's Format

#### *5.1.1 Tuples Collected (c-Tuples): from Secure Devices to the Recipient*

During the collection phase, the collected tuples ($c$-tuples for short) flow from secure devices to the recipient with the objective of allowing the recipient to build the dataset and to produce the sanitization information. In the rest of the paper, we call $\mathcal{T}^c \leftarrow \{t^c\}$ the set of $c$-tuples collected.

*Generic Format* A $c$-tuple $t^c$ originating from a secure device first contains the participant's record $r$, fully obfuscated through the use of a randomized encryption scheme (e.g., the `AES` algorithm used in `CBC` mode). The part of the $c$-tuple containing the encrypted record is noted $t^c.e$: $t^c.e \leftarrow \mathtt{E}_k(r)$, where $\mathtt{E}$ is a randomized encryption scheme, $k$ is the secure device's set of cryptographic keys, and $r$ the raw record.

No information leaks from the encrypted record, precluding the recipient to perform any (useful) computation on it. Consequently, in order to enable its participation in the construction phase, collected tuples must additionally embed a controlled amount of information about the raw record; we call it the *construction information* and denote it $t^c.c$. Obviously, the nature of the construction information depends on $\mathbb{A}$, the PPDP algorithm being implemented. We denote $\mathtt{F}_{\mathbb{A}} : dom \rightarrow img_{\mathtt{F}_{\mathbb{A}}}$, the function extracting the construction information from each record. Choosing the appropriate construction information results from a tension between the algorithm-dependant computation to be performed by the recipient and the disclosure tolerated for enabling it. We will further discuss this in Section 6.1.1 and assume for the moment that sufficient and harmless construction information is defined for the Met$_{\mathbb{A}}$P instance of the given PPDP algorithm.

Finally, each $c$-tuple contains *security information* dedicated to preventing attacks from an adversarial recipient. We will clearly define it in Section 6 and limit ourselves here to its notation $t^c.\zeta$.

As a result, $c$-tuples originating from secure devices follow the format:

$$t^c \leftarrow (e, c, \zeta) \text{ with } t^c.e \leftarrow \mathtt{E}_k(r) \text{ and } t^c.c \leftarrow \mathtt{F}_{\mathbb{A}}(r)$$

*Example:* $\mathbb{A} \leftarrow$ *$\alpha\beta$-Algorithm* In the $\alpha\beta$-Algorithm for $(d,\gamma)$-Privacy, the construction information must allow the recipient to identify fake records that are duplicates (of true or fake ones). First, revealing the true or fake nature of records lets the recipient distinguish between true and fake tuples[17]. Second, revealing the *equality relationship* between records through a deterministic *Message Authentication Code*[18] [21] (`MAC` for short) is sufficient for identifying duplicates. As discussed in Section 6.1.1, this twofold construction information neither leaks information about the content of the raw record nor leads to statistical attacks (recall that any true record is unique in the $\alpha\beta$-Algorithm specific data settings)). We consequently set the extraction function to return a pair made of: a bit indicating whether the record is true or fake (see Footnote 17) and a deterministic `MAC`: $\mathtt{F}_{\alpha\beta}(r) \leftarrow (\mathtt{True}(r)?, \mathtt{MAC}^d_k(r))$, where $k$ is the secure device's set of cryptographic keys and $r$ is a raw record.

*Example:* $\mathbb{A} \leftarrow$ *Bucketization* In the Bucketization algorithm for $\ell$-Diversity, the construction information must allow the recipient to assign each record to the bucket that corresponds to the record's sensitive data. Thus, the raw sensitive data is clearly sufficient for this purpose and does not provide the recipient with any information beyond the output of the Bucketization algorithm. We consequently simply set the construction information of a record $r$ to be $r$'s cleartext sensitive value: $\mathtt{F}_{Buck}(r) \leftarrow r.SD$.

[17] Revealing the true or fake nature of a record does not put privacy in danger except if the recipient is able to link it to the corresponding decrypted record. At this point of the protocol, it is linked to the encrypted record only; Section 6.1.2 will focus on attacks that aim at linking it to its corresponding decrypted record.

[18] Informally speaking, a MAC can be seen as a cryptographic hash whose output depends on a secret key.

*Example:* $\mathbb{A} \leftarrow$ *Mondrian* Finally, in MONDRIAN for $k$-ANONYMITY, the construction information must allow the recipient to recursive split the quasi-identifier space. We chose here to reveal the raw quasi-identifiers to the recipient: such construction information is sufficient for MONDRIAN and is tolerable in many real-case settings. Our straightforward implementation thus simply sets the construction information of a record $r$ to be $r$'s cleartext quasi-identifier: $\mathtt{F}_{Mond}(r) \leftarrow r.QID$.

### 5.1.2 Tuples to be Sanitized (s-Tuples): from the Recipient to Secure Devices

The construction phase is in charge of producing the set of tuples to be sanitized, called *s-tuples* and denoted $\mathcal{T}^s \leftarrow \{t^s\}$, that flow from the recipient to secure devices in order to be sanitized.

*Generic Format* A $s$-tuple $t^s$ flowing from the recipient to a secure device to be sanitized must contain both an encrypted record $t^s.e \leftarrow t^c.e$, where $t^c \in \mathcal{T}^c$, and the sanitization information $t^s.s$ needed to sanitize it. The nature of the sanitization information depends on the computation performed by the recipient over the construction information. Let $\mathtt{C}_{\mathbb{A}} : \{img_{\mathtt{F}_{\mathbb{A}}}\}^* \rightarrow img_{\mathtt{C}_{\mathbb{A}}}$ denote the *construction function* in charge of performing the global algorithm-dependant analysis of the construction information and $\mathtt{M} : img_{\mathtt{C}_{\mathbb{A}}} \times img_{\mathtt{F}_{\mathbb{A}}} \rightarrow img_{\mathtt{M}_{\mathbb{A}}}$ denote the mapping function in charge of mapping to each construction information its corresponding sanitization information (the input/output domains of both functions are algorithm-dependant). The resulting $s$-tuples follow the format:

$$t^s \leftarrow (e, s, \zeta), \text{ with } t^s.e \leftarrow \mathtt{E}_k(r) \text{ and } t^s.s \leftarrow \mathtt{M}_{\mathbb{A}}(\mathtt{C}_{\mathbb{A}}(\{\mathtt{F}_{\mathbb{A}}(r)\}), \mathtt{F}_{\mathbb{A}}(r))$$

*Example:* $\mathbb{A} \leftarrow$ *$\alpha\beta$-Algorithm* In the $\alpha\beta$-ALGORITHM, the sanitization information has to indicate whether the associated record is true or fake in order to let secure devices know whether a record should be kept with certainty or not. The construction function $\mathtt{C}_{\alpha\beta}$ thus removes the fake tuples that are duplicates (those that are not duplicates are kept). The mapping function $\mathtt{M}_{\alpha\beta}$ thus maps each record's $\mathtt{MAC}$ to 1 if the record is true, and 0 otherwise.

*Example:* $\mathbb{A} \leftarrow$ *Bucketization* In the BUCKETIZATION algorithm, the sanitization information has to indicate the equivalence class to which the sensitive data of the associated record belongs. The construction function $\mathtt{C}_{Buck}$ thus constructs the equivalence classes based on the sensitive data, and the mapping function $\mathtt{M}_{Buck}$ maps the sensitive data of each record to the identifier of its equivalence class.

*Example:* $\mathbb{A} \leftarrow$ *Mondrian* The construction function $\mathtt{C}_{Mond}$ builds the equivalence classes based on the quasi-identifiers. The MONDRIAN mapping function $\mathtt{M}_{Mond}$ is threfore similar to its BUCKETIZATION counterpart except that the input domain is the domain of quasi-identifiers instead of the domain of sensitive data.

| Cryptographic Schemes Used in this Article | |
|---|---|
| $\mathtt{E}$ | Randomized Encryption Scheme |
| $\mathtt{MAC}$ | Randomized MAC Scheme |
| $\mathtt{MAC}^d$ | Deterministic MAC Scheme |

| Functions Defined in this Article | |
|---|---|
| $\mathtt{F}_{\mathbb{A}} : dom \rightarrow img_{\mathtt{F}_{\mathbb{A}}}$ | Extraction Function |
| $\mathtt{C}_{\mathbb{A}} : \{img_{\mathtt{F}_{\mathbb{A}}}\}^* \rightarrow img_{\mathtt{C}_{\mathbb{A}}}$ | Construction Function |
| $\mathtt{M}_{\mathbb{A}} : img_{\mathtt{C}_{\mathbb{A}}} \times img_{\mathtt{F}_{\mathbb{A}}} \rightarrow img_{\mathtt{M}_{\mathbb{A}}}$ | Mapping Function |
| $\mathtt{S}_{\mathbb{A}} : dom \times img_{\mathtt{M}_{\mathbb{A}}} \rightarrow dom_{\mathcal{T}^r}$ | Sanitization Function |

| Data Structures | |
|---|---|
| $\mathcal{T}^c \leftarrow \{t^c\}$ | Set of Tuples Collected |
| $t^c.e \leftarrow \mathtt{E}_k(r)$ | Encrypted Record of $t^c \in \mathcal{T}^c$ |
| $t^c.c \leftarrow \mathtt{F}_{\mathbb{A}}(r)$ | Construction Information of $t^c \in \mathcal{T}^c$ |
| $t^c.\zeta$ | Security Information of $t^c \in \mathcal{T}^c$ (see Sec. 6) |
| $\mathcal{T}^s \leftarrow \{t^s\}$ | Set of Tuples to be Sanitized |
| $\mathcal{T}^s_i \subset \mathcal{T}^s$ | Partition $i$ of $\mathcal{T}^s$ |
| $t^s.e \leftarrow \mathtt{E}_k(r)$ | Encrypted Record of $t^s \in \mathcal{T}^s$ |
| $t^s.s \leftarrow \mathtt{M}_{\mathbb{A}}(\mathtt{C}_{\mathbb{A}}(\{\mathtt{F}_{\mathbb{A}}(r)\}), \mathtt{F}_{\mathbb{A}}(r))$ | Sanitization Information of $t^s \in \mathcal{T}^s$ |
| $\mathcal{T}^r \leftarrow \{t^r\}$ | Set of Sanitized Tuples |
| $t^r \leftarrow \mathtt{S}(t^s)$ | Sanitized Tuple |

**Table 1** Notations

### *5.1.3 Resulting Sanitized Tuples (r-Tuples): from Secure Devices to the Recipient*

Finally, sanitized tuples flow from secure devices to the recipient, letting him build the sanitized release. We denote the resulting set of (now sanitized) tuples $\mathcal{T}^r \leftarrow \{t^r\}$ and call them *r-tuples*. Each $r$-tuple is the result of applying the algorithm-dependant sanitization function to the decrypted record and sanitization information of an $s$-tuple: $\mathtt{S}_{\mathbb{A}} : dom \times img_{\mathtt{M}_{\mathbb{A}}} \rightarrow dom_{\mathcal{T}^r}$, where *dom* is the domain of raw records, $img_{\mathtt{M}_{\mathbb{A}}}$ is the algorithm-dependant domain of sanitization information, and $dom_{\mathcal{T}^r}$ is the algorithm-dependant domain of sanitized $r$-tuples. As a result:

$$t^r \leftarrow \mathtt{S}(t^s.e, t^s.s), \text{ with } t^s \in \mathcal{T}^s$$

## 5.2 Correct Execution Sequence

We now present the abstract phases that compose MET$_{\mathbb{A}}$P in Section 5.2.1, how they are instantiated on the three running example algorithms in Section 5.2.2, and finally in Section 5.2.3 we give some examples of other models that are supported.

### *5.2.1 Abstract Phases*

To summarize, any global publishing algorithm instantiating MET$_{\mathbb{A}}$P will follow the execution sequence described below and depicted in Figure 7.

During the collection phase, the recipient gathers the set of $c$-tuples $\mathcal{T}^c$ from the secure devices that connect. As stated above, each $c$-tuple contains the encrypted participant's record together with its corresponding construction information and security information. Similarly to traditional statistical surveys, the recipient stops the collection phase and launches the construction phase once it has collected enough $c$-tuples with respect to its needs.

The construction phase consists in letting the publisher produce the set of $s$-tuples to be sanitized $\mathcal{T}^s$ based on the $c$-tuples collected. It computes the sanitization information from the construction information previously collected, and forms $\mathcal{T}^s$ by appending to each encrypted record its corresponding sanitization information. Finally, it partitions $\mathcal{T}^s$ uniformly such that each partition $\mathcal{T}_i^s \in \mathcal{T}^s$ fits the limited resources of a single secure device.

Each secure device that connects during the sanitization phase downloads a self-contained partition $\mathcal{T}_i^s \in \mathcal{T}^s$, and uploads on the recipient the result of decrypting and sanitizing the $s$-tuples it contains. The recipient then stores the sanitized $r$-tuples in $\mathcal{T}^r$ and stops the sanitization phase once all the partitions have been sanitized.

### *5.2.2 Examples of Instances*

We illustrate below the correct (but unprotected) execution sequence of MET$_{\mathbb{A}}$P based on the three running examples of the article.

*Example:* $\mathbb{A} \leftarrow \alpha\beta$*-Algorithm*

- **Collection phase**: Each secure device that connects sends to the recipient a set of $c$-tuples where each $c$-tuple is as follows: $t^c \leftarrow (\mathtt{E}_k(r), (\mathtt{True}(r)?, \mathtt{MAC}_k(r)), \emptyset)$, where $r$ is its (true) record. The recipient stores the tuples received in the set of collected tuples: $\mathcal{T}^c \leftarrow \{t^c\}$.
- **Construction phase**: The recipient identifies the fake records that are duplicates (of true or fake ones) based on their associated `MAC` and true/fake bit and removes them (construction function $\mathtt{C}_{\alpha\beta}$). Note that the fake records that are not duplicates are kept as required by the $\alpha\beta$-ALGORITHM. It then forms the set of tuples to be sanitized $\mathcal{T}^s$ by appending to each encrypted record both its true/fake bit (mapping function $\mathtt{M}_{\alpha\beta}$) and its security information (empty for the moment). Finally, it partitions $\mathcal{T}^s$ so that each partition fits the limited resources of a single secure device.

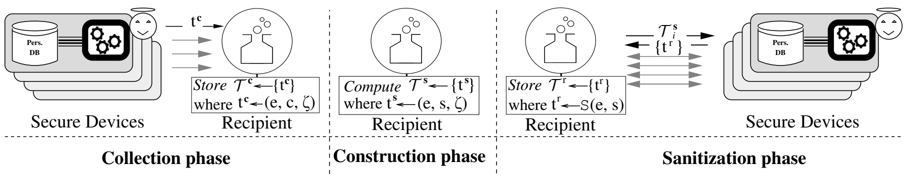

Pers. DB
$t^c$
Store $\mathcal{T}^c \leftarrow \{t^c\}$
where $t^c \leftarrow (e, c, \zeta)$
Secure Devices
Recipient
Compute $\mathcal{T}^s \leftarrow \{t^s\}$
where $t^s \leftarrow (e, s, \zeta)$
Recipient
$\mathcal{T}_i^s$
$\{t^r\}$
Store $\mathcal{T}^r \leftarrow \{t^r\}$
where $t^r \leftarrow S(e, s)$
Recipient
Secure Devices
Pers. DB
Collection phase
Construction phase
Sanitization phase

**Fig. 7** MET$_{\mathbb{A}}$P Execution Sequence

- **Sanitization phase**: Each secure device that connects is now able to sanitize a partition by downloading it and returning, decrypted, the records associated with bit 0 *with certainty* (i.e., the fake records) and those associated with bit 1 with probability $\alpha + \beta$ (i.e., the true records).

*Example:* $\mathbb{A} \leftarrow$ *Bucketization*

- **Collection phase**: Each secure device that connects sends to the recipient $(\mathtt{E}_k(r), r.SD, \emptyset)$, where $r$ is the secure device's record.
- **Construction phase**: The recipient executes the Bucketization algorithm over the set of raw sensitive data and obtains the resulting equivalence classes (construction function $\mathtt{C}_{Buck}$). It then forms the set of tuples to be sanitized $\mathcal{T}^s$ by associating each encrypted record to both the identifier of its equivalence class (mapping function $\mathtt{M}_{Buck}$) and its security information (still empty for the moment). Finally, it partitions $\mathcal{T}^s$ so that each partition fits the limited resources of a single secure device.
- **Sanitization phase**: Each secure device that connects is now able to sanitize a partition by downloading it and returning, decrypted, the quasi-identifier of each record with the identifier of its equivalence class.

*Example:* $\mathbb{A} \leftarrow$ *Mondrian*

- **Collection phase**: Each secure device that connects sends to the recipient $(\mathtt{E}_k(r), r.QID, \emptyset)$, where $r$ is the secure device's record.
- **Construction phase**: The recipient executes the Mondrian algorithm over the set of QIDs and obtains the resulting equivalence classes (construction function $\mathtt{C}_{Mond}$). It forms the set of tuples to be sanitized as above (using $\mathtt{M}_{Mond}$) and partitions it.
- **Sanitization phase**: Each secure device that connects is now able to sanitize a partition by downloading it and returning, decrypted, the records' sensitive data.

### *5.2.3 Supporting Various Privacy Models and Algorithms*

We show in Table 2 a sample of well-known models that can be enforced by our example algorithms based on possible construction and sanitization information. The table is divided in three parts. The first part synthesizes the instances of the running examples that are described thoroughly along the paper; we do not describe it further here.

The second part sketches possible straightforward uses of these instances for enforcing the well known $t$-Closeness [38] and $\epsilon$-Privacy [39] models. Both models belong to the partition-based approach and, similarly to $\ell$-Diversity, are based on constraining the distribution of sensitive data in each equivalence class as well as the number of quasi-identifiers. All these models can be directly enforced through our instance of the Bucketization algorithm (or its SABRE variant [11] for $t$-Closeness).

| **Model** | **Algorithm** | **Construction Info.** | **Sanitization Info.** |
|---|---|---|---|
| **Proposed Instances for the Running Examples** | | | |
| $(d,\gamma)$-Privacy (eq. Differential Privacy) | $\alpha\beta$-Algorithm | − True/fake bit<br>− Equality relationship between records | True/fake bit |
| $k$-Anonymity | Mondrian | Raw quasi-identifiers | Eq. class' ID |
| $\ell$-Diversity | Bucketization | Raw sensitive data | *idem* |
| **Straightforward Use of Proposed Instances** | | | |
| $t$-Closeness | SABRE | Raw sensitive data | Eq. class' ID |
| $\epsilon$-Privacy | Bucketization | *idem* | *idem* |
| **Prospective Instances of Running Examples** | | | |
| $k$-Anonymity | Mondrian | Order relationship between quasi-identifiers' attributes | Eq. class' ID |
| $\ell$-Diversity | *idem* | − Order relationship between quasi-identifiers' attributes<br>− Raw sensitive data | *idem* |
| $t$-Closeness | *idem* | *idem* | *idem* |
| $\epsilon$-Privacy | *idem* | *idem* | *idem* |

**Table 2** Coverage of the Running Examples (Sample)

Finally, the third part shows the above sample of models enforced by an instance of Mondrian for which the construction information about quasi-identifiers is limited to the *order-relationship* between their attributes. Indeed, on the one hand, for enforcing these models Mondrian needs information on both the sensitive data and the quasi-identifiers, but on the other hand, the precise association between the two parts must be obfuscated. Limiting the disclosure about quasi-identifiers to their order-relationship, which is sufficient information for Mondrian, may solve this dilemma. However, while the first and second parts can be used as-is, the third part remains prospective. The investigation of *order-preserving encryption schemes* is indeed at an early stage [10] - their practical use is conditioned on a better understanding of their security implications.

Table 2 serves an illustrative purpose: the models it contains are only a sample of well-known models enforced by the algorithms used in this paper as running examples. Despite its non-exhaustivity, this table sketches how representative models of the two major PPDP approaches can be enforced on MetAP. Designing algorithms specifically for MetAP and exploring the profusion of already existing PPDP algorithms is left open. To these ends, *property-preserving encryption schemes* [45], that aim at revealing a precise property of the data while keeping the rest obfuscated, will be of particular interest for defining construction information that is both sufficient and secure (security is discussed in the next section), the cornerstone of MetAP instances.

### 5.3 Correctness of $\text{M}\textsc{et}_{\mathbb{A}}\text{P}$

In this section, we have presented the execution sequence of $\text{M}\textsc{et}_{\mathbb{A}}\text{P}$. We state the following correctness theorem[19] which shows that it is possible to correctly conduct a PPDP algorithm $\mathbb{A}$ on an Asymmetric architecture using $\text{M}\textsc{et}_{\mathbb{A}}\text{P}$.

**Theorem 1 (Met$_{\mathbb{A}}$P Correctness)** *Let $\pi$ be an instance of a privacy algorithm $\mathbb{A}$ instantiated on an* Asymmetric *architecture using* $\text{M}\textsc{et}_{\mathbb{A}}\text{P}$. *If the construction information is sufficient to perform the construction phase in a centralized setting, then $\pi$ is* correct *and* terminates.

*Proof (Correctness (sketch))* The recipient gathers the participant's records (in the form of $c$-tuples) during the collection phase. This essentially amounts to sampling the set of secure devices similarly to a trusted server context. The construction phase then starts and, provided that the construction information is sufficient, computes the sanitization information and produces the set of tuples to be sanitized. Finally, the sanitization phase makes use of *any* connected secure device to sanitize it partition by partition. The information contained in each partition is sufficient for its sanitization because a partition includes both the encrypted records to sanitize and their corresponding sanitization information. As a result, since each phase is similar to its centralized counterpart, $\pi$ is *correct.* Provided that the size of each partition fits the resources of a secure device, and because secure devices connect indefinitely, $\pi$ also *terminates.* □

## 6 Securing the Meta-Protocol

The outcome of the preceding section is the design of a correct execution sequence for $\text{M}\textsc{et}_{\mathbb{A}}\text{P}$ suitable for the Asymmetric architecture. In this section, we consider an adversarial recipient that seeks to breach the privacy of participants (see Section 2.1.3 for the definition of the attack models). We base the design of our security measures on (1) a thorough analysis of the attack vectors available to the recipient, (2) an identification of the abstract *safety properties* to be asserted (by trusted devices) during the execution sequence, and (3) implementations of these safety properties that show their feasibility on an Asymmetric architecture. This section is organized by increasing adversarial strength. Section 6.1 explains how to prevent passive attacks (protection against an honest-but-curious recipient). Section 6.2 analyzes the possible actions a malicious (active) recipient can carry out and explains how to detect them, thereby negating the possibility of an active attack. It then explores in Section 6.3 detection and therefore deterrence of a $Malicious_{Hard}$ recipient, i.e., *having breached one or more secure devices* and using their cryptographic keys to forge tuples. Finally, Section 6.4 describes a methodology to implement any PPDP algorithm using one of the proposed $\text{M}\textsc{et}_{\mathbb{A}}\text{P}$ protocols.

[19] Due to space reasons, in this paper we present sketches of proofs. The complete proofs of all theorems can be found in [2].

## 6.1 Honest-but-Curious Recipient

The honest-but-curious recipient is a passive adversary. It aims at breaching the privacy of records by launching inferences on the information that it obtains from secure devices, explicitly or not. We start off in Section 6.1.1 by discussing the information explicitly revealed to the recipient in order to correctly execute MET$_{\mathbb{A}}$P, i.e., the tuples (collected and sanitized) sent by secure devices. Then, we tackle in Section 6.1.2 the inferences made possible by the information that is not explicitly revealed by MET$_{\mathbb{A}}$P but that leaks from its execution sequence. Indeed, due to the "per-partition" organization of the correct but unprotected execution sequence, the recipient is able to trivially *link* the sanitized records corresponding to a given partition to the sanitization information corresponding to the same partition. Consequently, it may also link it to the corresponding construction information. Such *linking attacks* may lead to the full disclosure of records. We define the UNLINKABILITY safety property to counter this attack, and sketch a feasible implementation[20]. Finally, in Section 6.1.3, we insert the assertion of the UNLINKABILITY property into MET$_{\mathbb{A}}$P's execution sequence, making it robust against an honest-but-curious recipient.

### *6.1.1 Information Explicitly Revealed*

During the execution of MET$_{\mathbb{A}}$P, the recipient obtains from secure devices two data structures: the set of collected *c*-tuples $\mathcal{T}^c$ (where each of these *c*-tuples contains an encrypted record together with its construction information and its security information) and the set of sanitized tuples $\mathcal{T}^r$.

As stated earlier, the construction information delivered to the recipient in the *c*-tuples must be *harmless* in the sense that it must not increase the recipient's knowledge beyond what it already knew or what will be revealed through the sanitized release. There can be three different cases:

– Case 1: The construction information provides no information at all;
– Case 2: The construction information provides information that can be deduced from $\mathcal{T}^r$;
– Case 3: The construction information provides information that cannot be deduced from $\mathcal{T}^r$;

In the first two cases, the information provided is harmless. In the third case, the results of MET$_{\mathbb{A}}$P hold if this construction information can be shown to be harmless *given the adversarial knowledge.* The truthfullness of such a hypothesis depends on each specific adversary and data setting. Let us illustrate each of these cases through the three algorithms used as running examples.

[20] Due to space reasons, in this paper and in Appendix A, we present sketches of the safety propreties' implementation. The complete description of all implementations can be found in [2].

*Example: $\alpha\beta$-Algorithm (Case 1)* The construction information of a record is a pair made of a bit indicating the true/fake nature of the record and the record's deterministic `MAC` (i.e., $t^c.c \leftarrow$ ($\texttt{True}(r)?$, $\texttt{MAC}_k(r)$) for a given record $r$). First, the true/fake bit leaks nothing about the record's actual content (the $\alpha\beta$-Algorithm is precisely based on the indistinguishability between true and fake records). Second, since a `MAC` is indistinguishable from a random bitstring, it does not leak anything neither about the raw record. Finally, since the $(d, \gamma)$-Privacy's specific context assumes that each true record is unique, the set of `MAC`s follows a uniform distribution completely uninformative to the adversary: frequency analysis is useless. As a result, an adversarial recipient does not learn *anything* based on the construction information used in our implementatin of $\alpha\beta$-Algorithm.

*Example: Bucketization (Case 2)* The construction information of a record is its cleartext sensitive data (i.e., $t^c.c \leftarrow r.SD$ for a given record $r$). Since the cleartext sensitive data is part of the $\ell$-diverse equivalence classes finally released, an adversarial Recipient does not learn *anything beyond the final output* of the Bucketization algorithm.

*Example: Mondrian (Case 3)* The construction information of a record is its quasi-identifier (i.e., $t^c.c \leftarrow r.QID$ for a given record $r$). While this does not thwart the $k$-Anonymity privacy model (the link between a given quasi-identifier and its corresponding sensitive data remains ambiguous), the recipient obtains *slightly more information* than the coarsened quasi-identifiers of the $k$-anonymous equivalence classes output by Mondrian (Case 3)[21]. In many realistic settings, it is common to assume that the adversary knows (at least a subset of) the raw quasi-identifiers (e.g., after accessing the list of participants in an opinion's survey). If this assumption is met, the $\text{Met}_{\mathbb{A}}\text{P}$ can be instantiated as Mondrian by disclosing the quasi-identifiers. If it is not, either some different construction information must be found (as discussed in Section 5.2.3, only the order relationship between the quasi-identifiers' attributes is necessary for Mondrian), or another algorithm enforcing $k$-Anonymity must be used. Finally, it is easy to observe that *no information explicitly revealed jeopardizes the records' privacy*, neither through the collected tuples nor through the sanitized tuples:

- $t^c.e$, the encrypted record (part of a collected tuple): does not leak any information about the raw record;
- $t^c.c$, the construction information (part of a collected tuple): does not increase the attacker's knowledge as discussed above;
- $t^c.\zeta$, the security information (part of a collected tuple): it is used to detect malicious recipients so its content will be made clear in the corresponding

[21] Our implementation of Mondrian publishes the sanitized release in a form equivalent to a release made of two tables where the one contains raw quasi-identifiers, the other contains sensitive data, and both can be (lossy) joined through a class' identifier. This form of publishing releases, called Anatomy, was proposed in [49] to increase their utility.

section (Section 6.2) - we require *by definition* that it does not reveal anything about the raw record (e.g., it can be encrypted using a randomized scheme);

– $t^r$, a sanitized tuple: by definition the sanitized tuples satisfy the chosen PPDP model and are harmless;

### *6.1.2 Linking Attacks*

Since the sanitization is performed on a *per-partition* basis in order to fit the modest resources of secure devices, the recipient knows the link between each sanitized partition (containing a subset of sanitized tuples) and its corresponding non-sanitized partition (containing a subset of sanitization information and linked to a subset of construction information). Indeed, it is important to note that the recipient has access to $\mathcal{T}^c$, $\mathcal{T}^s$ and $\mathcal{T}^r$, and builds itself $\mathcal{T}^s$ from $\mathcal{T}^c$: it knows which subsets of $\mathcal{T}^c$ and $\mathcal{T}^s$ correspond to a given subset of $\mathcal{T}^r$. For example, within the $\alpha\beta$-Algorithm, the recipient links: (1) a subset of decrypted records in $\mathcal{T}^r$, sent by secure devices in response to the reception of a partition in $\mathcal{T}^s$, to (2) the bitmap indicating for each record whether it is true or not and contained in the partition. The privacy breach is blatant when all records of the partition are fake (resp. true).

In order to prevent such inference, we need to destroy the link between subsets of sanitized tuples in $\mathcal{T}^r$ and subsets of sanitization information in $\mathcal{T}^s$. The `Unlinkability` safety property, defined below, asserts that the mapping between each $r$-tuple and any subset of $s$-tuples is as probable with and without the information obtained due to the partitioning scheme.

**Definition 1 (`Unlinkability` Safety Property)** Let $\Psi_1$ denote the data structure mapping each partition to the $s$-tuples it contains, and $\Psi_2$ the data structure mapping each partition to its corresponding $r$-tuples. Let $\leftrightarrow$ be the binary operator denoting the possible mapping between a $r$-tuple and (a subset of) $s$-tuples. The set of sanitized tuples satisfies `Unlinkability` if:

$$\forall t^r \in \mathcal{T}^r\ \forall t^s \subset \mathcal{T}^s, \mathtt{Pr}[t^r \leftrightarrow t^s | \Psi_1, \Psi_2] = \mathtt{Pr}[t^r \leftrightarrow t^s]$$

The linking attack exploits all the data structures obtained by the publisher (i.e., $\mathcal{T}^c$, $\mathcal{T}^s$, and $\mathcal{T}^r$) and the information leaking from the execution sequence (i.e., the partitioning of $\mathcal{T}^s$ and $\mathcal{T}^r$). No other information (explicitly revealed or not) is available to the recipient to draw other inferences. As a result, the `Unlinkability` safety property is sufficient to render harmless the information that is not explicitly revealed to the recipient by secure devices.

*Implementation Sketch* The implementation of `Unlinkability` consists in letting secure devices return sanitized records in an encrypted form (consequently yielding $\mathcal{T}^{r,o}$) and shuffle them before disclosing them to the recipient. The shuffling is performed incrementally. The recipient starts by splitting each partition of encrypted sanitized records into $n$ parts, called *buckets*. Any secure

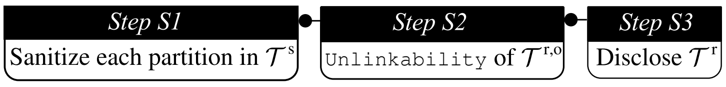

Step S1
Sanitize each partition in $\mathcal{T}^{s}$
Step S2
Unlinkability of $\mathcal{T}^{r,o}$
Step S3
Disclose $\mathcal{T}^{r}$

**Fig. 8** $\textsc{Met}_{\mathbb{A}}\textsc{P}_{(hc)}$: Sanitization Phase

device is able to pick $n$ buckets coming from $n$ distinct partitions, and return them shuffled. The shuffling circuit is organized such that the first round shuffles the initial partitions $n$ by $n$ (each record returned is equally likely to originate from any of $n$ partitions), the second round shuffles the result of the first round $n$ by $n$ (each record returned is equally likely to originate from any of $n^2$ partitions), and so on until any encrypted record is equally likely to originate from any partition. They are finally decrypted. Note that during the shuffling, sanitized records are re-encrypted by a randomized encryption scheme before being returned to the recipient: the latter must not know to which downloaded record a record returned corresponds to.

### *6.1.3 A Correct and Secure Protocol Protecting Against Honest-but-Curious Recipients*

The information disclosed explicitly is guaranteed to be harmless by definition and the effects of the information that is not revealed explicitly are countered by the `Unlinkability` safety property. As depicted in Figure 8 and described in Algorithm 1, the execution sequence of the $\textsc{Met}_{\mathbb{A}}\textsc{P}_{(hc)}$'s sanitization phase consists of three steps, thereby enhancing $\textsc{Met}_{\mathbb{A}}\textsc{P}$ (we omit the collection and construction phases: they remain identical):

- **S1**: secure devices download the set of tuples to be sanitized partition per partition, and return to the recipient sanitized tuples *in an encrypted form*;
- **S2**: they enforce the `Unlinkability` safety property on the set of encrypted sanitized tuples, thereby destroying the link between each subset of encrypted sanitized tuples and its corresponding subset of sanitization information;
- **S3**: they finally decrypt the sanitized tuples.

**Theorem 2** *Let $\pi_{(hc)}$ be the instance of a privacy algorithm $\mathbb{A}$ instantiated on an* ASYMMETRIC *architecture using $\textsc{Met}_{\mathbb{A}}\textsc{P}_{(hc)}$ (Algorithm 1). If the construction information is sufficient to perform the construction phase in a centralized setting and does not independantly provide any information to the recipient, then $\pi_{(hc)}$ correctly and securely computes $\mathbb{A}$ against a honest-but-curious recipient.*

*Proof (sketch)*

- The `Unlinkability` property has no impact on the correctness of the sequence, therefore given Theorem 1, $\textsc{Met}_{\mathbb{A}}\textsc{P}_{(hc)}$'s execution sequence is correct and terminates.

```
Algorithm 1: MET_A P_(hc) - Execution Sequence
   req. (recipient)        : Privacy param. (algorithm dependant)
   req. (secure devices): The cryptographic keys $\kappa$, privacy param. (algorithm
         dependant).
1  begin collection phase
2      The recipient receives c-tuples of the form $t^c \leftarrow (o, c, \zeta)$, where $t^c.e \leftarrow \mathtt{E}_\kappa(r)$,
       $t^c.c \leftarrow \mathtt{F}_{\mathbb{A}}(r)$ is algorithm dependant, and $t^c.\zeta \leftarrow \emptyset$ is the security information (it
       will be defined in the version of the protocol that copes with a malicious
       recipient). Let $\mathcal{T}^c \leftarrow \cup t^c$;
3      When $\mathcal{T}^c$ meets the required cardinality, the recipient switches to the
       construction phase;
4  begin construction phase
5      The recipient generates the set of tuples to be sanitized: $\forall t^c \in \mathcal{T}^c$ output
       $t^s \leftarrow \mathtt{M}_{\mathbb{A}}(\mathtt{C}_{\mathbb{A}}(\{\mathcal{T}^c.c\}), t^c.c)$ and store them in $\mathcal{T}^s$: $\mathcal{T}^s \leftarrow \{t^s\}$;
6      The recipient partitions $\mathcal{T}^s$;
7  begin sanitization phase
8      forall the $\mathcal{T}_i^s \in \mathcal{T}^s$ do
9          Send $\mathcal{T}_i^s$ to a connecting secure device and store the encrypted sanitized
           records in $\mathcal{T}^{r,o}$: $\mathcal{T}^{r,o} \leftarrow \mathcal{T}^{r,o} \cup \mathtt{Sanitize}(\mathcal{T}_i^s)$;
10     Secure devices apply Unlinkability to the encrypted sanitized records and
       decrypt them: $\mathcal{T}^r \leftarrow \mathtt{Unlinkability}(\mathcal{T}^{r,o})$;
```

– An honest-but-curious recipient obtains the set of collected tuples $\mathcal{T}^c$, the set of tuples to be sanitized $\mathcal{T}^s$, and the sanitized release $\mathcal{T}^r$. By hypothesis, the knowledge of $\mathcal{T}^c$, $\mathcal{T}^s$, and $\mathcal{T}^r$ independently from each other does not bring any information to the adversary. Second, any attack based on the joint analysis of $\mathcal{T}^c$, $\mathcal{T}^s$, and $\mathcal{T}^r$ consists in mapping the non-encrypted information contained in all: construction information to sanitization information, and then sanitization information to their corresponding sanitized record. This second mapping step is prohibited by the `Unlinkability` safety property. As a result, the honest-but-curious recipient does not learn anything more than it already knew, or that it could deduce from the sanitized release. $\pi_{(hc)}$ is therefore secure.

In consequence $\pi_{(hc)}$ correctly and securely computes $\mathbb{A}$ against a honest-but-curious recipient. □

It is important to note that the instance of any algorithm under MET$_{\mathbb{A}}$P$_{(hc)}$ only requires to show that the construction information is harmless: if this is the case, Theorem 2 can consequently be used to prove that the global execution sequence is secure.

## 6.2 Malicious Recipient

In this section, we study the case of a malicious recipient. The approach is based on the identification of a set of safety properties that must be enforced in order to prevent active attacks. In Section 6.2.1, we give an intuitive idea of these properties and in Section 6.2.2 we describe the proposed algorithm. Since there are many technicalities in the formal description of safety properties, we have chosen not to discuss them in the core of the paper, and instead we refer the reader interested in their full details to Appendix A.

### *6.2.1 Overview of the Approach*

A malicious attacker is not limited to passive attacks but may deviate arbitrarily from the protocol. As stated in Section 2.1.3, the objectives of the malicious recipient are to conduct such an attack that will increase its knowledge, while generating a correct result and remaining undected by secure devices. It can attack the integrity of all data structures it can access in the course of the protocol (i.e. the collected dataset and the sanitization information) or try to tamper the execution sequence itself. This leads to three forms of active attacks:

– **Tamper the set of tuples to be sanitized:** without loss of generality, the recipient can tamper $\mathcal{T}^s$ by *forging*, *copying*, or *deleting* $s$-tuples (with respect to data authentication, modifying an existing $s$-tuple is similar to forging a new $s$-tuple): secure devices must therefore *check the integrity of the set of s-tuples to be sanitized.* This can be done by enforcing the following safety properties. First, `Origin` prevents forge actions by checking the authenticity of each $c$-tuple based on a cryptographic signature. Second, `Identifier Unicity`, `Mutual Exclusion`, and `Membership` prevent copy actions both within each partition and accross the set of partitions based on a unique identifier assigned to each tuple by secure devices. Finally, the `Invariance` property checks that the set of tuples to be sanitized remains unmodified during the course of MET$_{\mathbb{A}}$P, thus preventing delete actions. These safety properties are formally defined in Appendix A.1, and shown to protect against the aforementionned malicious attacks (through Lemma 1, for the protection against copy actions (involving several properties), and directly, for the protections against forge and delete actions).
– **Tamper the construction function:** because the construction function is delegated to the recipient, a malicious recipient can simply tamper its execution (e.g., within the $\alpha\beta$-ALGORITHM, removing too many fake records so that the privacy guarantees do not hold): secure devices must then assert the `Safe Construction Function` safety property. Despite its high algorithm-dependant nature, verifying the safety of the construction function amounts in most cases to *counting* some values (e.g., number of fake tuples for the $\alpha\beta$-ALGORITHM, number of tuples per equivalence class for MONDRIAN). We thus propose the `Secure Count` primitive that lets secure devices compute these counts in a secure way. The overall methodology is described thoroughly in Appendix A.2.
– **Tamper the execution sequence:** the attacker may disorganize the execution of the protocol, e.g., by obtaining the sanitized tuples before having made them unlinkable: secure devices must finally assert the `Execution Sequence Integrity` safety property in order to guarantee both the

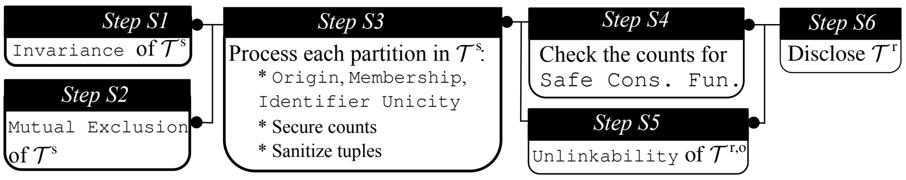

Step S1
Invariance of $\mathcal{T}^{s}$
Step S2
Mutual Exclusion of $\mathcal{T}^{s}$
Step S3
Process each partition in $\mathcal{T}^{s}$:
* Origin, Membership, Identifier Unicity
* Secure counts
* Sanitize tuples
Step S4
Check the counts for Safe Cons. Fun.
Step S5
Unlinkability of $\mathcal{T}^{r,o}$
Step S6
Disclose $\mathcal{T}^{r}$

**Fig. 9** MET$_{\mathbb{A}}$P$_{(mal)}$: Sanitization Phase

completeness of the execution sequence and its order. We detail in Appendix A.3 how we *chain* the execution sequence through cryptographic signatures to this end.

As a result, by guaranteeing that any possible active attack will be detected, safety properties force an active malicious adversary to be passive (a case covered in Section 6.1), for which we have already proposed a correct and secure protocol.

### *6.2.2 A Correct and Secure Protocol Against Malicious Recipients*

We depict in Figure 9 the execution sequence of MET$_{\mathbb{A}}$P$_{(mal)}$'s sanitization phase (detailed in Appendix A.3).

**Theorem 3** *Let $\pi_{(mal)}$ be the instance of the privacy algorithm $\mathbb{A}$ under* MET$_{\mathbb{A}}$P$_{(mal)}$. *If the information disclosed by the construction information does not independantly provide any information to the recipient, and if* `Execution Sequence Integrity` *is enforced, then $\pi_{(mal)}$ correctly and securely computes $\mathbb{A}$ against a malicious recipient with weakly-malicious intents.*

*Proof (Sketch)* The proof comes in three parts.

– First of all, the `Origin`, `Identifier Unicity`, `Mutual Exclusion`, `Membership`, and `Invariance` safety properties (presented in Appendix A.1) preclude any malicious action over the set of tuples to be sanitized. Moreover, the `Safe Construction Function` safety property (presented in Appendix A.2) guarantees the legitimacy of the sanitization information produced by the recipient. Then the execution sequence of MET$_{\mathbb{A}}$P$_{(mal)}$ is correct and secure against a malicious attacker.
– Second, since the `Execution Sequence Integrity` property is guaranteed, these safety properties cannot be circumvented (MET$_{\mathbb{A}}$P$_{(mal)}$ is executed completely and in the correct order): the malicious attacker cannot conduct any active attack.
– Finally, since the malicious recipient is reduced to honest-but-curious behaviours then MET$_{\mathbb{A}}$P$_{(mal)}$ respects the execution sequence of MET$_{\mathbb{A}}$P$_{(hc)}$, so that Theorem 2 lets us conclude that $\pi_{(mal)}$, being an instance of MET$_{\mathbb{A}}$P$_{(mal)}$, is correct and secure against a malicious recipient.

□

**Algorithm 2:** $\textsc{Met}_{\mathbb{A}}\textsc{P}_{(mal)}$

```
req. (recipient) : The partitioned set of tuples to be sanitized ($\mathcal{T}^s$), the set of
    labels for data structures to be made invariant ($L$), the number of secure
    devices in the designated population ($n_{pop}$).
req. (secure devices): The cryptographic keys $\kappa$, the privacy parameters
    (algorithm dependant), the number of secure devices in the designated
    population ($n_{pop}$).
1  The recipient computes $\mathcal{T}^s$'s summary $\Sigma_{\mathcal{T}^s}$ which maps each partition's TID-Set
   (expressed as a range) to its digest. Let $\Sigma_{\mathcal{T}^s}.\Theta$ denote the list of partitions's ranges
   (ordered), and $\Sigma_{\mathcal{T}^s}(T)$ denote the digest to which the range $T \in \Sigma_{\mathcal{T}^s}.\Theta$ maps;
2  Step S1: Guarantee Invariance of $\mathcal{T}^s$ : $p_{inv}$ ←Invariance($L.\mathcal{T}^s$, $\Sigma_{\mathcal{T}^s}$)
3  Step S2: Check Mutual Exclusion of $\mathcal{T}^s$: $p_{mutex}$ ←Mutual Exclusion($\Sigma_{\mathcal{T}^s}$, $p_{inv}$,
   $L.\mathcal{T}^s$)
4  begin Step S3: Process each partition in $\mathcal{T}^s$
5      forall the $\mathcal{T}_i^s \in \mathcal{T}^s$ do
6          The recipient sends $\mathcal{T}_i^s$, $\Sigma_{\mathcal{T}^s}$, and $p_{mutex}$ to a connecting secure device,
           which performs the Sanitize function and returns both $\mathcal{T}_i^s$ sanitized and
           obfuscated ($\mathcal{T}_i^{r,o}$) ←Sanitize($\mathcal{T}_i^s$, $\Sigma_{\mathcal{T}^s}$, $p_{mutex}$)) and the secure counts for
           the Safe Construction Function related to $\mathcal{T}_i^s$;
7          The recipient gathers the counts of the partitions Counts;
8          The recipient adds the obfuscated sanitized partition to the obfuscated
           sanitized release: $\mathcal{T}^{r,o} \leftarrow \mathcal{T}^{r,o} \cup \mathcal{T}_i^{r,o}$
9  Step S4: Check the counts for Safe Construction Function based on the Counts
   data structure;
10 Steps S5 and S6: Perform Unlinkability and disclosure of $\mathcal{T}^r$.
```

## 6.3 Coping with a Malicious$_{Hard}$ Recipient

In the $\textsc{Met}_{\mathbb{A}}\textsc{P}_{(mal)}$ protocol, if the adversarial recipient succeeds in breaking at least one secure device, it can unveil not only the devices' information but also its cryptographic keys which can in turn be used to decrypt the complete set of encrypted records. To limit the scope of such attacks, the traditional solution is to use clusters, each cluster using its own key, and organize the process so that the impact of compromising one key is divided by the number of clusters. We follow this approach and partition the devices into a set of $n_c$ distinct *clusters*, *randomly* and *evenly*, such that the secure devices belonging to different clusters are equipped with distinct cryptographic keys. Breaking a secure device consequently brings the adversary the ability to decrypt the encrypted records from the device's cluster only. This security measure requires a few adjustments to $\textsc{Met}_{\mathbb{A}}\textsc{P}$ in order to handle the limited decryption abilities of secure devices. Moreover, in addition to decrypting data, the compromised device gains the capacity of encrypting and signing data (e.g., $s$-tuples, secure counts), allowing it to satisfy the `Origin` safety property. $\textsc{Met}_{\mathbb{A}}\textsc{P}$ must also be protected against this kind of forge actions.

We start by giving an example of an attack that can be launched by a $Malicious_{Hard}$ attacker. We then describe the adjustments of $\textsc{Met}_{\mathbb{A}}\textsc{P}$ to cope with the limited decryption abilities of secure devices. We introduce `Typicallity`, the new safety property in charge of limiting the forge ability of a

$Malicious_{Hard}$ recipient, and conclude by describing the practical situation where a cluster has been broken and the breach has been detected.

### *6.3.1 A $Malicious_{Hard}$ attack example*

Let us consider a $k$-anonymous release of 1.000.000 participants with $k = 100$, divided into $n_c = 500$ clusters, each cluster containing therefore $n_p = 2000$ participants. Breaking a secure device yields the possibility to retrieve the information of $n_p$ participants. However, the attacker can now generate signed $s$-tuples. If the attacker wishes to attack a participant of a given QID, the attacker need only generate $k - 1 = 99$ tuples with the same QID. By looking at the results of the equivalence class with this QID, it can determine what the value of the sensitive data of this participant is. If the attacker can generate as many forged $s$-tuples as it wants, then it can reconstruct the entire dataset. Our goal is therefore to limit the number of $s$-tuples the attacker can forge, in order to limit the impact of the attack, via the detection of this attack. We show in Section 7.3 that the detection probability of such an attack is close to 1.

### *6.3.2 Managing Clustered Cryptographic Keys*

Clusters introduce a new challenge to MET$_{\mathbb{A}}$P. It is possible to imagine two different approaches to be used in order to manage a PPDP process spanning $n_c$ clusters : (i) run $n_c$ independent PPDP processes, one per cluster, without changing anything to the implemented protocol or (ii) adapt the implementations of the properties whose inputs and/or outputs span multiple clusters.

The advantage of the (straightforward) solution (i) is that no change to the implementation is needed. The only constraint is that cluster size must be chosen large enough to yield acceptable quality of the privacy metric that is used (e.g. choosing clusters whose size is much greater than $k$ in the case of $k$ anonymity). However, the problem is that the output of the $n_c$ processes will *not* be exactly the same as the output of a single, centralized process: e.g. in the case of the MONDRIAN algorithm, the equivalence classes produced by each cluster are computed on a dataset whose size depends on the cluster size, introducing a trade-off between the utility of the release and the limitation of the scope of a potential attack.

Solution (ii) is more complex in terms of implementation, but yields a release equivalent to the non clustered protocol, which is one of the requirements of our approach. Fortunately, a large number of safety properties' implementations can be used with no change. `Invariance`, `Mutual Exlusion`, `Origin`, `Identifier Unicity`, `Membership`, and `Execution Sequence Integrity` fall in this category. On the contrary, the implementation of `Unlinkability` (that shuffles all $s$-tuples together whatever their originating cluster) and `Safe Construction Function` (the `Secure Count` primitive computes global counts over the full set of $s$-tuples) need a specific adaptation to clusters. As shown in Appendix B, the adaptation consists in a few minor additions to their non-clustered implementations. Consequently, the clustered execution sequence is mostly performed independently by each cluster based on the standard non-clustered implementations. Overall protocol latency will however be higher, depending now on the number of clusters, as shown by the experiments in Section 7.1.2.

### *6.3.3 Protection Against Forge actions*

The `Origin` safety property is no longer sufficient to preclude forge actions by a $Malicious_{Hard}$ recipient. Indeed, such actions cannot be detected by cryptographic means because tuples produced by the compromised device are as legitimate as tuples produced by honest secure devices. Forge attacks consist in overwhelming the tuples collected from secure devices with forged tuples (e.g., within $(d,\gamma)$-PRIVACY, the recipient can forge fake $s$-tuples easily recognizable and remove them from the sanitized release at the end). However, in order to be effective, the tampered cluster will be generating far more tuples than a legitimate cluster. This is both the strength of the attack (it reduces the effective privacy parameter proportionally to the number of fake tuples generated) and its weakness (it is easier to detect). The problem of detecting such attacks is similar to the traditional *input falsification* detection problem [12] where compromised data sources produce wrong data. Here again we follow the traditional approach by detecting clusters that produce more tuples than the others based on probabilistic outlier detection techniques (e.g., [8]). The new `Typicality` safety property, defined in Appendix B and implemented based on `Secure Count`, formalizes this requirement. It is integrated in the execution sequence in a similar way to the production and checking of counts for the `Safe Construction Function`: the required counts are produced during step S3 and summed up and checked before the final decryption of step S6.

### *6.3.4 Post attack actions*

We now investigate what can be done, once the attack is detected. There can be two cases concerning the previous attacks : either (i) the attacker colludes with the recipient, or (ii) the attacker is a participant that hopes to gain subsequent access to the release (e.g. because it is a public survey) and infer information by exploiting the forged $s$-tuples (and therefore $r$-tuples) it contains.

Case (ii) is trivial, since this attack will be detected by the recipient during the collection phase. The recipient can chose to merely discard the whole cluster, with no impact on the privacy of participants. This case is therefore a DoS type of attack, where no knowledge can be gained by the attacker.

In case (i), the attack is revealed by participating secure devices during the sanitization phase, indicating that the recipient was aware of the abnormal distribution, and is thereby convicted. In such a case, participants in the cluster will have to exhibit their unaltered secure device in order to prove their innocence (e.g. physical tampering of secure devices is easily visible). New keys can then be redistributed to all the righteous owners of unaltered secure devices in this cluster for future studies.

In both cases, attack detection is sufficient to deter a $Malicious_{Hard}$ attacker, since its attack will be detected, and the attacker itself will be convicted.

### 6.4 Methodology to Instantiate MET$_{\mathbb{A}}$P$_{(mal)}$ on a Specific Algorithm

Instanciating MET$_{\mathbb{A}}$P$_{(mal)}$ on a specific algorithm is quite straightforward, once the algorithm has been instanciated using MET$_{\mathbb{A}}$P$_{(hc)}$. The only algorithm-dependant processes are the following. The designer of the algorithm must:

1. Analyse the construction information and prove that an attacker will learn nothing from it;
2. Provide an implementation of the `Safe Construction Function` property. As shown in Appendix A.2, the `Secure Count` primitive is a powerful building block that appears to be sufficient for most implementations of the `Safe Construction Information` property;

If this is the case, then Theorem 3 proves that the execution of MET$_{\mathbb{A}}$P$_{(mal)}$ is correct and secure. It is important to note that a designer who would want to create from scratch a PPDP protocol using secure devices would need to prove the two above points, *plus* show that the global protocol is correct and secure. This would be done in a similar manner as the one we propose: protecting against attacks that link cyphered inputs to cleartext outputs, that endanger the integrity of the dataset or of the execution sequence, and that produce unsafe sanitization information. Our contribution is to reduce to the strict minimum the algorithm-dependant analysis, and provide generic security guarantees.

## 7 Experimental Validation

This section validates experimentally the feasibility of our approach. It studies MET$_{\mathbb{A}}$P's costs first in terms of *latency* (i.e., the number of secure device connections required to perform each execution step) and second in terms of *internal time consumption* (i.e., the time spent within a secure device to perform each unitary task), and it finally shows the effectiveness of a simple implementation of the `Typicality` safety property in thwarting forge-based attacks. Although the experimental results are obtained through the $\alpha\beta$-ALGORITHM instance of MET$_{\mathbb{A}}$P, the behaviour observed is general and does not depend on any specific algorithm. MET$_{\mathbb{A}}$P has been fully implemented and we are currently integrating it in the PlugDB prototype [6]. PlugDB is a personal data server embedded in a portable secure device of the type *smart USB token* (thoroughly described below); it is currently experimented in the French Yvelines District to implement medical-social folders facilitating home care for elderly people. The smart token we are using has the following characteristics: it has a USB-key form factor, the microcontroller is equipped with a 32 bit RISC CPU clocked at 50MHz, a crypto-coprocessor implementing AES and SHA in hardware, 64 KB of RAM, 1 MB of NOR-Flash and is connected to a 1GB external NAND-Flash; the token communicates with the terminal with full speed USB throughput. This smart token is provided by Gemalto, the industrial partner of the project and world-leader in smart-cards. Unfortunately, we cannot change the code embedded in this pre-industrial platform so our measurements have been performed on a cycle-accurate hardware emulator of this same token. The latency results were obtained through simulation, since this factor is highly application dependent.

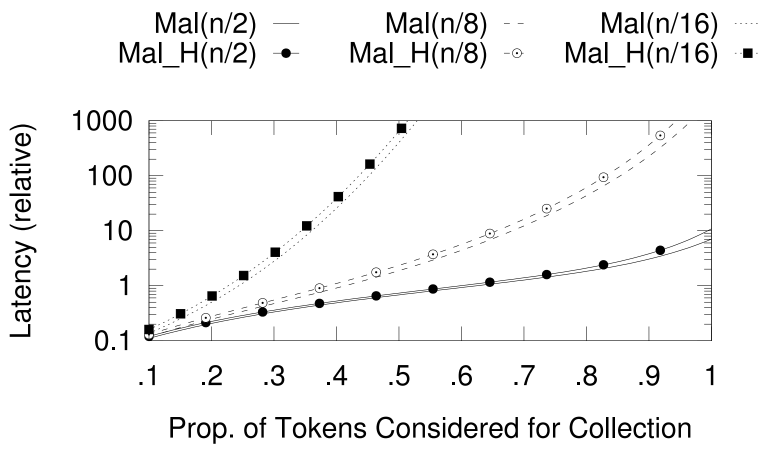
## 7.1 Latency

We model the latency of the collection and sanitization phases in terms of number of token connections required to perform them (the latency on the recipient side is negligible compared to them). In what follows, we note $n_\theta$ the number of tokens (participants), $\gamma \in [0; 1]$ the proportion of tokens to be collected ($\gamma n_\theta$ is therefore the number of participants), and $n_c$ the number of clusters, when applicable ($Malicious_{Hard}$ model).

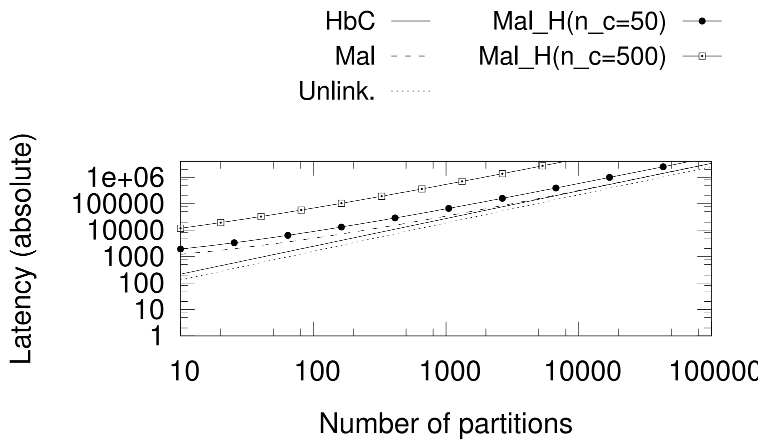
The collection phase's latency depends on *the percentage of distinct tokens* that are required to connect in order to build the initial dataset; measuring it obviously assumes a specific distribution of tokens' connection probabilities. In the honest-but-curious or malicious (without hardware attack) attack model, since *any* connecting token can participate in the sanitization process, this phase's latency does not depend on *which* specific token connects; it only depends on *the number of partitions* in the partitioned dataset, and the number of tokens connected simultaneously. In the $Malicious_{Hard}$ model, it is possible to adopt two approaches for the collection phase: (i) globally collect $\gamma n_\theta$ $c$-tuples or (ii) collect exactly $\gamma n_\theta / n_c$ $c$-tuples from each cluster. If approach (i) is used, then the latency will be exactly the same as in a non-clustered context. If approach (ii) is used, there is a slight overhead. Results are presented in Section 7.1.1.

The sanitization phase makes no assumption on the distribution of token connections. We simply compute the global number of (non distinct) connections required. In the $Malicious_{Hard}$ model, the sanitization phase is divided into two parts : (i) the execution of $n_c$ phases (in parallel) on $n_c$ times less values using $n_c$ times less tokens and (ii) a inter-cluster computation. Phase (i) presents a small overhead compared to the non-clustered case (results are similar to the collection phase cluster overhead), phase (ii) presents an overhead that is proportional to the number of clusters. Results are presented in Section 7.1.2.

In what follows, we study the latencies of the collection and sanitization phases according to the parameter they respectively depend on. Further conversions of the latency into any time unit can be obtained by modeling the



(a) Collection Phase



(b) Sanitization Phase

**Fig. 10** Latencies

application-dependent time variations of tokens's connections (through token assignment probability, presented in Section 7.1.1).

### *7.1.1 Collection Phase*

*Setting* To make the presentation more general, we do not use token connection probabilities, but token assignment probabilities, which means we consider all token connections to the recipient as sequential. For each connection, we define an assignment function (we use a parametrized Gaussian distribution) which returns the token connected. We add parallelism by defining the length of a *step* as a number of simultaneous connections, polled from the sequential connection stream. In our experiments, we measure the latency as the total number of connections of non-necessarily distinct tokens. This number can easily be converted into a temporal value, as we show below.

Formally, we represent the population of tokens by a set of integers, each token corresponding to an integer between 1 and $n_\theta$. We use a Gaussian distribution $f(x, \mu, \sigma)$ whose expectation $\mu$ is fixed to the halfway integer, i.e., $n_\theta/2$, and whose standard deviation $\sigma$ is a variable parameter to represent the assignment probability, i.e. that token $x$ is chosen for a given connection. For each connection step, we randomly sample $n_{conn}$ (e.g. $n_{conn} = 0.01 \times n_\theta$) non-necessarily distinct values from this distribution to determine the set of tokens connected. We simulate different applicative behaviors by varying the standard deviation. Although this model does not capture an application where half the tokens ($n_1 = 0.5$) connect very often with probability $p_1$, while the other half ($n_2 = 0.5$) connect rarely with probability $p_2$, it captures a worse case of this model where half the tokens connect with probability $p'_1 \leq p_1$ and the other half connect with probability $p'_2 \leq p_2$, and is therefore able to provide an upper bound on the temporal latency of such an application.

*Results* Figure 10(a) shows the latency of the collection phase according to the percentage of distinct tokens required to connect in order to build the initial dataset. In this figure, the latency is expressed as the proportion of (non-necessarily distinct) token connections having occurred with respect to the total number of tokens, and we call it *relative latency*. We plot the relative latencies corresponding to three distributions of token assignment probabilities characterized by various standard deviations (i.e., $\sigma = n_\theta/2$, $\sigma = n_\theta/8$, and $\sigma = n_\theta/16$). Obviously the relative latency becomes exponential if one wants to poll the *whole* population for low $\sigma$ values.

This latency is exactly the same in a clustered protocol in which the participation of each cluster is random. As stated in Appendix B, outlier detection is much simpler if all clusters have exactly the same proportion of participating tokens. In this case, the latency of the protocol is the latency of the slowest of all $n_c$ clusters. Figure 10(a) illustrates the latency of a clustered protocol with this constraint (the curves called Mal_H), with the hypothesis that the clusters all exhibit the same connection behaviour. As we can see, the overhead is low (around 10% extra duration on average).

Note that in order to estimate the real *temporal latency* of the protocol, we need to multiply the relative latency by the length of each connection step (e.g., 1 minute) and divide by the proportion of tokens connected during that step (e.g., 1% of total number of tokens). For instance with $\sigma = n_\theta/2$ and a proportion collected of 50% of the population then the relative latency read in figure 10(a) is 0.7. The global latency is therefore $0.7 \times 1 \times 10^2 = 70$ min. Note that the temporal latency depends on the proportion of tokens connecting at each step, and not the total number of tokens, provided the recipient can accept enough incoming connections. This means that given $\sigma$ the characteristic of token assignment probability, and given a certain time limit, we can fix the maximum proportion of the population that can be polled within that time limit, thus ensuring scalability.

#### *7.1.2 Sanitization Phase*

The latency of the sanitization phase is free from the assumption on the distribution of tokens' connections. It is expressed as the absolute number of (non distinct) required token connections, assuming that each token remains connected long enough to execute all its tasks (approximately 10 seconds at most, see below).

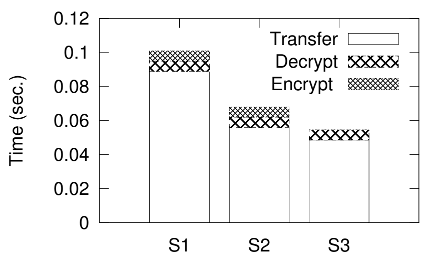
Figure 10(b) shows the linear evolution of the sanitization phase latency according to the number of partitions (independently of partition's size). The latencies of both the honest-but-curious (HbC) and malicious (Mal) protocols are similar because primarily made of the implementation of the `Unlinkability` safety property, whose cost dominates the others. The lower bound latency on these protocols' sanitization phase corresponds to the time taken by one token to process a partition, assuming that enough tokens can connect simultaneously. The latency of the $Malicious_{Hard}$ (Mal_H) protocol was obtained for two numbers of clusters ($n_c = 50$ and $n_c = 500$). It is larger than the other latencies due to its inter-cluster `Unlinkability` implementation that essentially requires each partition to be downloaded by each cluster and is consequently proportional to the number of clusters times the number of partitions (see Appendix B for more details).

### 7.2 Internal Time Consumption

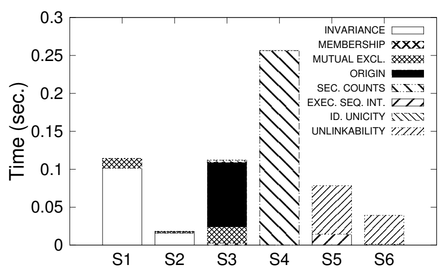
*Setting (Dataset)* The data schema is an extract of the CENSUS[22] dataset's schema often used in PPDP works; it consists of attributes {*Age (78 distinct values), Education (17), Marital (6), Race (9), Work-Class (10), Country (83), Salary-Class (50)*}. The domain of each attribute has been mapped to the domain of integers. Indeed, neither the MET$_{\mathbb{A}}$P nor the $\alpha\beta$-ALGORITHM are sensitive to the actual values of collected $c$-tuples; only *the number of partitions* and *the size of a partition* have an impact. The dataset contains approximately 1 million true records. The privacy parameters of the $\alpha\beta$-ALGORITHM were chosen following the authors of [47] (their exact values do not matter here) and resulted in the insertion of approximately 1.5 million random records.

Figure 11(a) depicts the maximum internal time consumption of $\alpha\beta$-ALGORITHM$_{(hc)}$'s execution steps (see Section 6.1), grouped by basic operation, each partition containing 1K $s$-tuples (regardless of the number of partitions). Except for steps S2 and S3 (`Unlinkability` enforcement), the cost is due to running the protocol without any safety property. We call it the basic cost. The total cost remains under one second and is mainly composed of the transfer cost, confirming the affordable costs of cryptographic operations.

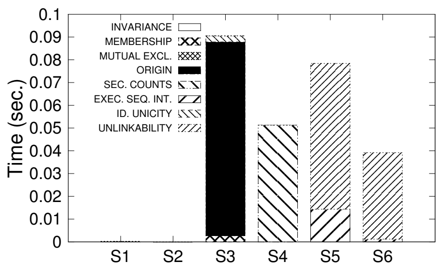
[22] `http://ipums.org/`



(a) Times Honest-but-Curious



(b) Malicious

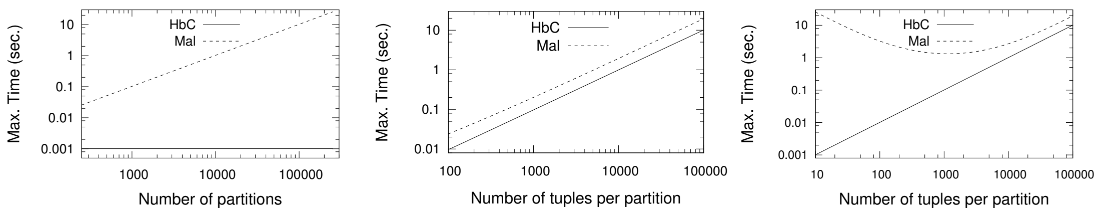


(c) Malicious$_{Hard}$

**Fig. 11** Internal Time Consumption: Details (2.5M Tup.: 1K Tup./Part., 2.5K Part.)

Figure 11(b) plots the maximum internal time consumption of the $\alpha\beta$-Algorithm$_{(mal)}$'s main execution steps, grouped by safety property, for 2.5K partitions in the partitioned dataset, each containing 1K $s$-tuples. We obtained the measures based on a single token. We observe that the safety properties overhead is primarily due to `Invariance` during Step S1, and the counts of `Safe Construction Function` during Step S4. The other safety properties exhibit negligible costs.

We plot the internal time consumption of the $Malicious_{Hard}$ version of the protocol in Figure 11(c) (with $n_c = 500$ clusters). The number of partitions considered by a token is smaller in the $Malicious_{Hard}$ version because the dataset is distributed among clusters: the cost of the properties concerning the set of partitions (i.e., `Invariance` and `Mutual Exclusion`) is consequently also divided by the number of clusters. The `Secure Count` performed in step S4 for the `Safe Construction Function` also exhibits a smaller cost. Indeed, in its $Malicious_{Hard}$ version, the tokens download a number of counts proportional to the number of clusters and not to the number of partitions anymore (i.e., the outputs of the intra-cluster shufflings). The implementation of `Typicality` based on the `Secure Count` primitive yields the same cost. The cost of the other safety properties depends on the number of tuples per partition (e.g., `Origin`, `Identifier Unicity`, `Membership`). As a result, the most costly safety properties have an internal time consumption reduced in



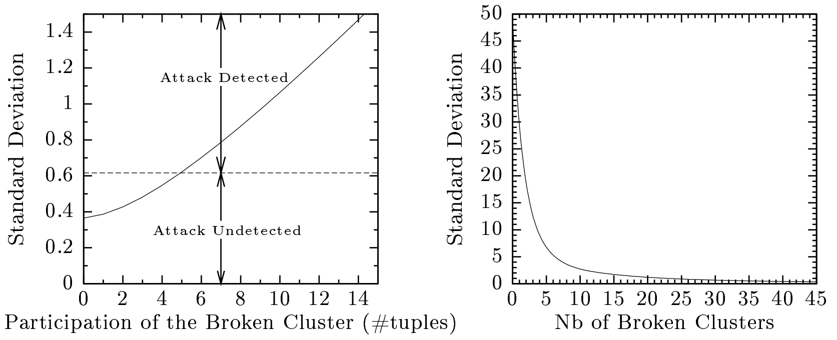
(a) Varying the number of Part. (10 Tup./Part.) (b) Varying the number of Tup./Part. (10 Part.) (c) Varying both (Total 1M Tup.)

**Fig. 12** Internal Time Consumption: Scaling

the $Malicious_{Hard}$ version. Nevertheless, there will be more devices involved in the protocol, so the overall computation time will be greater.

Figure 12(a) shows the variation of the *maximum* internal time consumptions of the honest-but-curious and malicious versions of the protocol with respect to the total number of partitions (the size of a partition being fixed to a negligible quantity - 10 $s$-tuples): while the honest-but-curious version of the protocol (HbC) remains unimpacted by these variations, the cost of the malicious safety properties (Mal) varies linearly according to the linear increase in size of the data structures made invariant and of the number of counts to sum up to check the rules. Figure 12(b) shows the same cost but depending on the partition's size (the total number of partitions being fixed to 10). It highlights the linear nature of the basic cost and the light overhead of the malicious safety properties that depend on the partition's size. Finally, Figure 12(c) considers a fixed dataset of 1 million $s$-tuples and shows the variations of the cost according to both the (inter-related) size of a partition and number of partitions. It outlines on the malicious curve, the point at which the costs that depend on the number of partitions (i.e., handling the summary) becomes negligible with respect to the costs that depend on the partition's size.

We can draw the following conclusions from this performance analysis. First of all, the critical aspect seems to be the latency of the collection phase. This latency is indeed not bounded and highly application dependent. It is determined by the connection rate of participants (e.g., in the medical field this rate can vary from a few connection per year up to several connections per day for chronic diseases) and by the ratio of the whole population to be polled. Indeed, according to statisticians and epidemiologists, collecting data is always the critical step and the right metric is better expressed in weeks or months than in seconds, and in fact, the use of connected secure devices could even speed up this process, traditionally performed manually. The interesting point is that contrary to a manual approach, with MET$_{\mathbb{A}}$P the latency is not linked to the size of the population. Second, regarding internal time consumption, experiments show that MET$_{\mathbb{A}}$P can manage very large datasets with excellent performance due to its parallelism and to the hardware implementation of cryptographic operations on smart tokens. Hence, scalability is achieved.



(a) Standard Deviation Sensitivity wrt the Participation of the Broken Cluster (b) Standard Deviation Sensitivity wrt the number of Broken Clusters

**Fig. 13** Outlier Detection

## 7.3 Detecting Forge Actions

We now turn our focus to the effectiveness of the `Typicality` property in thwarting attacks based on forge actions. In this section, we demonstrate experimentally the efficiency of the attack detection, without any predefined constraint on the number of $s$-tuples per cluster, through a straightforward analysis of the standard deviation of the number of $s$-tuples per cluster. Though more complex analysis could be designed by, e.g, using various statistical measures depending on the participations distribution (e.g., measures used for outliers detection [8]), our results show that the standard deviation analysis already reaches high detection rates despite its simplicity.

We consider a dataset made of $|\mathcal{T}| = 10^6$ $s$-tuples, sent by participants partitioned uniformly at random in $|\Omega| = 5.10^2$ clusters. In our experiments, all clusters are of equal size, but comparable results would be achieved with a normal distribution of participants in clusters. Since the distribution of secure devices to clusters is random, the number of $s$-tuples per cluster in the dataset follows a normal distribution. We illustrate the impact of an attack on our simple $k$-Anonymity running example. The most powerful attack is the one that concentrates the $s$-tuples forged in a single equivalence class in order to disclose a target $s$-tuple: if the attacker forges $t_{forged}$ $s$-tuples, then the class under attack will contain $k - t_{forged}$ $s$-tuples. To test the typicality of a cluster $C_j \in \Omega$, we compute $\sigma$ the standard deviation of the number of $s$-tuples per cluster for each class (excluding clusters that do not participate in the class). In the general case, where $|\Omega| \geq k$ and there are no forged $s$-tuples, $\sigma$ is very small (in our experiment, its average value was $\sigma_{avg} \approx 0.36$ and its largest observed value was $\sigma_{max} \approx 0.62$).

Figure 13(a) shows the evolution of $\sigma$ function of the number of $s$-tuples forged by an attacking recipient having broken a single secure device (therefore a single cluster). In order to achieve perfect knowledge of a target $s$-tuple (e.g., the $s$-tuple of a target participant identified by its $QID$), the attacker would need to inject $k-1$ forged $s$-tuples in the target class. As shown in Figure 13(a), a cluster participating more than 5 $s$-tuples leads to a statistically improbable value (i.e., $\sigma \gg \sigma_{max}$). For example, for a realistic value of $k$, e.g., $k = 100$, the attacker must inject 99 forged $s$-tuples to completely thwart $k$-Anonymity, but is limited to a negligible quantity. Note that Figure 13(a) is a zoom: evolution is not linear but polynomial. If the attacker succeeds in breaking several clusters (meaning breaking secure devices from different clusters), forged $s$-tuples have less impact on the standard deviation because the attacker can distribute the participation over them. Figure 13(b) illustrates the value of $\sigma$ function of the number of broken clusters $b$ over which the attacker distributed evenly $k-1 = 99$ forged $s$-tuples (which means identifying the value of the only one that was not forged): at least 31 different clusters need to be broken to have $\sigma < \sigma_{max}$ and 43 to have $\sigma < \sigma_{avg}$. Situations that demand stronger detection levels can be satisfied simply by increasing the number of clusters. Indeed, it is obvious that the values of the standard deviations (average and maximal) are inversely proportional to the number of clusters.

Although more complex statistical analysis could be used (e.g., combining several statistical measures, chosing the measure according to the participations distribution), the above experimental results show that even a simple analysis of the standard deviation already makes $Malicious_{Hard}$ attacks harmless. Indeed, launching a successful attack would require breaking a large number of clusters, which is unrealistic because of the added costs of physically being in possession of a large number of secure devices and compromizing their hardware protections.

In consequence, checking the `Typicality` property suffices to deter a malicious adversary that has broken a secure device, and wishes to acquire more information, while remaining undetected.

## 8 Conclusion

The implementation and architectural facets of PPDP are often disregarded by the research community while they may lead to spectacular privacy breaches. This paper tries to build an alternative to the usual trusted publisher assumption. It capitalizes on the emergence of secure devices to revisit the PPDP problem. It proposes Met$_{\mathbb{A}}$P, a generic, distributed and scalable meta-protocol as the basis of a methodology to implement various PPDP algorithms. The article contains three theorems that can be used to prove that a given implementation of this protocol is correct (Th. 1) and secure when facing honest-but-curious (Th. 2) or malicious recipients (Th. 3). Beyond secure tokens, we believe that the proposed framework has the potential to match any context where participants a priori trust each other, whatever the source of this trust (tamper-resistance, certified code, large scale distribution over personal plug-computers, membership of a same community of users, etc). The key requirement is that cheating actions can still be detected by typicality tests even if the trust assumption turns out to be violated by a few participants.

We expect that the approach we have proposed with MET$_{\mathbb{A}}$P can pave the way to future studies trying to instantiate other privacy-aware algorithms (e.g., interactive queries) on an ASYMMETRIC architecture, while providing similar correctness and security guarantees.

## References

1. S. Agrawal and J. R. Haritsa. A framework for high-accuracy privacy-preserving mining. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 193–204, Washington, DC, USA, 2005. IEEE Computer Society.
2. T. Allard. *Sanitizing Microdata Without Leak: A Decentralized Approach.* PhD thesis, University of Versailles, 2011.
3. 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. *Proc. VLDB Endow.*, 3:25–35, September 2010.
4. T. Allard, B. Nguyen, and P. Pucheral. Safe Realization of the Generalization Privacy Mechanism. In *Proceedings of the 9th international conference on Privacy Security and Trust*, PST '11, pages 16–23, 2011.
5. T. Allard, B. Nguyen, and P. Pucheral. Sanitizing microdata without leak: combining preventive and curative actions. In *Proceedings of the 7th international conference on Information security practice and experience*, ISPEC'11, pages 333–342, Berlin, Heidelberg, 2011. Springer-Verlag.
6. N. Anciaux, L. Bouganim, Y. Guo, P. Pucheral, J.-J. Vandewalle, and S. Yin. Pluggable personal data servers. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 1235–1238, New York, NY, USA, 2010. ACM.
7. S. Bajaj and R. Sion. Trusteddb: a trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 205–216, New York, NY, USA, 2011. ACM.
8. V. Barnett and T. Lewis. *Outliers in Statistical Data.* Wiley, 3rd edition, 1994.
9. A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 128–138, New York, NY, USA, 2005. ACM.
10. A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 578–595, Berlin, Heidelberg, 2011. Springer-Verlag.
11. J. Cao, P. Karras, P. Kalnis, and K.-L. Tan. SABRE: a Sensitive Attribute Bucketization and REdistribution framework for t-closeness. *The VLDB Journal*, 20:59–81, February 2011.
12. H. Chan, H.-C. Hsiao, A. Perrig, and D. Song. Secure distributed data aggregation. *Found. Trends databases*, 3(3):149–201, Mar. 2011.
13. B.-C. Chen, D. Kifer, K. LeFevre, and A. Machanavajjhala. Privacy-preserving data publishing. *Found. Trends in Databases*, 2(1-2):1–167, January 2009.
14. G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 1253–1261, New York, NY, USA, 2011. ACM.
15. C. Dwork. Differential privacy. In *Proceeding of the 39th International Colloquium on Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2006.
16. Eurosmart. Smart USB Token (White Paper). Eurosmart, 2008.

17. M. Fischlin, B. Pinkas, A.-R. Sadeghi, T. Schneider, and I. Visconti. Secure set intersection with untrusted hardware tokens. In *Proceedings of the 11th international conference on Topics in cryptology: CT-RSA 2011*, CT-RSA'11, pages 1–16, Berlin, Heidelberg, 2011. Springer-Verlag.
18. B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42:14:1–14:53, June 2010.
19. G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast data anonymization with low information loss. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 758–769. VLDB Endowment, 2007.
20. Giesecke & Devrient. Portable security token. http://www.gd-sfs.com/portable-security-token. Last retrieved on the 27th of June 2012.
21. O. Goldreich. Foundations of cryptography: a primer. *Found. Trends in Theoretical Computer Science*, 1(1):1–116, 2005.
22. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.
23. L. A. Gordon, M. P. Loeb, W. Lucyshin, and R. Richardson. 2006 CSI/FBI Computer Crime and Security Survey. Computer Security Institute, 2006.
24. V. Goyal, Y. Ishai, M. Mahmoody, and A. Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 173–190. Springer Berlin / Heidelberg, 2010.
25. V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In D. Micciancio, editor, *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer Berlin / Heidelberg, 2010.
26. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 216–227, New York, NY, USA, 2002. ACM.
27. C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 491–500, New York, NY, USA, 2008. ACM.
28. IDC. IDC Defines the Personal Portable Security Device Market. http://tinyurl.com/IDC-PPSD. Last retrieved on the 27th of June 2012.
29. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded sfe: offloading server and network using hardware tokens. In *Proceedings of the 14th international conference on Financial Cryptography and Data Security*, FC'10, pages 207–221, Berlin, Heidelberg, 2010. Springer-Verlag.
30. W. Jiang and C. Clifton. A secure distributed framework for achieving k-anonymity. *The VLDB Journal*, 15:316–333, 2006.
31. P. Jurczyk and L. Xiong. Distributed anonymization: Achieving privacy for both data subjects and data providers. In *IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 191–207, Berlin, Heidelberg, 2009. Springer-Verlag.
32. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, EUROCRYPT '07, pages 115–128, Berlin, Heidelberg, 2007. Springer-Verlag.
33. D. Kifer. Attacks on privacy and definetti's theorem. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 127–138, New York, NY, USA, 2009. ACM.
34. D. Kifer and B.-R. Lin. Towards an axiomatization of statistical privacy and utility. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 147–158, New York, NY, USA, 2010. ACM.
35. D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 193–204, New York, NY, USA, 2011. ACM.

36. D. Kifer and A. Machanavajjhala. A rigorous and customizable framework for privacy. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 77–88, New York, NY, USA, 2012. ACM.
37. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 25–, Washington, DC, USA, 2006. IEEE Computer Society.
38. N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*, ICDE '07, pages 106–115, april 2007.
39. A. Machanavajjhala, J. Gehrke, and M. Götz. Data publishing against realistic adversaries. *Proc. VLDB Endow.*, 2(1):790–801, Aug. 2009.
40. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 24–, Washington, DC, USA, 2006. IEEE Computer Society.
41. A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '04, pages 223–228, New York, NY, USA, 2004. ACM.
42. N. Mohammed, R. Chen, B. C. Fung, and P. S. Yu. Differentially private data release for data mining. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 493–501, New York, NY, USA, 2011. ACM.
43. N. Mohammed, B. C. M. Fung, P. C. K. Hung, and C.-K. Lee. Centralized and distributed anonymization for high-dimensional healthcare data. *ACM Trans. Knowl. Discov. Data*, 4:18:1–18:33, October 2010.
44. N. Mohammed, B. C. M. Fung, K. Wang, and P. C. K. Hung. Privacy-preserving data mashup. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 228–239, New York, NY, USA, 2009. ACM.
45. O. Pandey and Y. Rouselakis. Property preserving symmetric encryption. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 375–391, Berlin, Heidelberg, 2012. Springer-Verlag.
46. V. Rastogi, D. Suciu, and S. Hong. The boundary between privacy and utility in data anonymization. *CoRR*, abs/cs/0612103, 2006.
47. V. Rastogi, D. Suciu, and S. Hong. The boundary between privacy and utility in data publishing. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 531–542. VLDB Endowment, 2007.
48. L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.
49. X. Xiao and Y. Tao. Anatomy: simple and effective privacy preservation. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 139–150. VLDB Endowment, 2006.
50. M. Xue, P. Papadimitriou, C. Raïssi, P. Kalnis, and H. K. Pung. Distributed privacy preserving data collection. In *Proceedings of the 16th international conference on Database systems for advanced applications - Volume Part I*, DASFAA'11, pages 93–107, Berlin, Heidelberg, 2011. Springer-Verlag.
51. A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.
52. N. Zhang and W. Zhao. Distributed privacy preserving information sharing. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 889–900. VLDB Endowment, 2005.
53. S. Zhong, Z. Yang, and T. Chen. k-anonymous data collection. *Inf. Sci.*, 179:2948–2963, August 2009.
54. S. Zhong, Z. Yang, and R. N. Wright. Privacy-enhancing k-anonymization of customer data. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 139–147, New York, NY, USA, 2005. ACM.

## A Malicious Recipient

### A.1 Protecting $\mathcal{T}^s$

*Forge Actions Precluded* The *forge* tampering action allows the attacker to forge tuples to be sanitized. The harmful effects of such attack are obvious; for example within the $\alpha\beta$-ALGORITHM, the recipient could forge all the fake records, launch the sanitization, and identify and remove them from the sanitized release. The `Origin` safety property (Definition 2) states that each collected tuple must be accompanied with a signature binding the encrypted record to its security information and guaranteeing their authenticity: the recipient is now unable to forge authentic $s$-tuples.

**Definition 2 (`Origin` Safety Property)** In order to respect the `Origin` safety property, each $c$-tuple embeds a signature (e.g. a randomized `MAC`), denoted $\sigma$, which is the result of signing the $c$-tuple's encrypted record concatenated to its security information: $\forall t^c \in \mathcal{T}^c$ then $t^c.\sigma \leftarrow \mathtt{MAC}(t^c.e||t^c.\zeta)$, where $||$ denotes the concatenation operator. Each $s$-tuple embeds the signature of its corresponding $c$-tuple as a proof of authenticity.

*Copy Actions Precluded* *Copy* actions allow the recipient to artificially increase the number of tuples to be sanitized in $\mathcal{T}^s$, and to identify their corresponding sanitized tuples in $\mathcal{T}^r$ based on the number of copies. For example within the $\alpha\beta$-ALGORITHM, the recipient may have stored a single fake tuple in $\mathcal{T}^s$, copy it $(n_{\mathcal{F}} - 1)$ times (where $n_{\mathcal{F}}$ denotes the number of fake records), launch the sanitization, and finally remove from the sanitized tuples $\mathcal{T}^r$ the ones that appear $n_{\mathcal{F}}$ times: only true tuples would remain.

Copy actions are actually twofold. With *intra-partition* copy actions, one or more $s$-tuples are copied several times into their own partition, while with *inter-partition* copy actions, the destination partition is different from the source partition. The safety properties in charge of detecting intra/inter-partition copy actions are based on(1) assigning to each tuple a unique identifier (thus making trivial the detection of duplicates in a partition), and(2) organizing the set of tuples to be sanitized such that each identifier is authorized to be part of a single partition only thus forcing the duplicates, if any, to be part of the same partition - where detection is trivial as stated in (1).The `Identifier Unicity` safety property (Definition 3) precludes intra-partition copy actions by requiring that in a given partition each tuple identifier be unique (see the implementation sketches below for an example of such identifier).

**Definition 3 (`Identifier Unicity` safety property)** Let $t^s \in \mathcal{T}_i^s$ be a $s$-tuple in the partition $\mathcal{T}_i^s$, and $t^s.\zeta.\theta$ denote $t^s$'s identifier. Partition $\mathcal{T}_i^s$ respects the `Identifier Unicity` safety property if for every pair of $s$-tuples $t_j^s, t_k^s \in \mathcal{T}_i^s$, $t_j^s.\zeta.\theta = t_k^s.\zeta.\theta \Rightarrow j = k$.

Inter-partition copies are more difficult to detect. In addition to being unique in its own partition, each $s$-tuple must only appear in a single partition. To this end, we define for each partition the set of identifiers it is *supposed* to contain, called the partition's TID-Set, and denote it $\mathcal{T}_i^s.\Theta$. The `Mutual Exclusion` safety property (Definition 4) ensures that no partition's TID-set overlaps, and the `Membership` safety property (Definition 5) that each identifier must appear in the partition to which it is supposed to belong. As a result, `Mutual Exclusion` and `Membership` together guarantee that each identifier actually appears within a single partition (as stated in Lemma 1).

**Definition 4 (`Mutual Exclusion` safety property)** Partitions respect `Mutual Exclusion` if for every pair of partitions $\mathcal{T}_i^s, \mathcal{T}_j^s \subset \mathcal{T}^s$, $i \neq j \Rightarrow \mathcal{T}_i^s.\Theta \cap \mathcal{T}_j^s.\Theta = \varnothing$.

**Definition 5 (`Membership` safety property)** A partition $\mathcal{T}_i^s$ respects `Membership` if for every $s$-tuple $t_j^s \in \mathcal{T}_i^s$, then $t_j^s.\zeta.\theta \in \mathcal{T}_i^s.\Theta$.

**Lemma 1** *Enforcing together the `Identifier Unicity`, `Mutual Exclusion`, and `Membership` safety properties is necessary and sufficient to guarantee the absence of any (intra/inter--partition) copy action.*

*Proof* We start by showing the sufficency of these properties. First, `Identifier Unicity` is sufficient to preclude by itself intra-partition copy actions (recall that the authenticity of a $s$-tuple and its identifier is guaranteed by the `Origin` safety property). Second, assume that a given $s$-tuple $t^s$ has been copied into two distinct partitions. Only the TID-Set of one of them contains $t^s$'s identifier because otherwise `Mutual Exclusion` would be contradicted. Consequently there must be one partition's TID-Set that does not contain $t^s$'s identifier. But this clearly contradicts the `Membership` safety property. As a result, `Membership` and `Mutual Exclusion` are together sufficient to preclude inter-partition copy actions.

We now show the necessity of these properties. First, since a distinct identifier is assigned to each $s$-tuple, the absence of intra-partition copy results immediately in the satisfaction of the `Identifier Unicity` property. Second, the absence of inter-partition copy implies that the partitioning is correct so that: (1) the `Mutual Exclusion` property is satisfied in that TID-Sets do not overlap (recall that a distinct identifier is assigned to each $s$-tuple) and(2) the `Membership` property too in that each $s$-tuple appears in the partition which TID-Set contains its identifier. □

*Delete Actions Precluded* The *delete* tampering action is based on sanitizing at least two versions of $\mathcal{T}^s$ where one version contains some tuples that do not appear in the other version (i.e., deleted). The deleted subset of $s$-tuples correspond to the subset of sanitized tuples that do not appear in $\mathcal{T}^r$ anymore. For example within Mondrian, the recipient could obtain the equivalence classes corresponding to a set of $s$-tuples (based on their quasi-identifiers), then obtain a second version of the classes where one tuple has been deleted; the sensitive value that does not appear anymore in $\mathcal{T}^r$ simply corresponds to the quasi-identifier of the deleted $s$-tuple.

Loosely speaking, the set of tuples to be sanitized is affected by a delete action if the set of tuples associated to a partition changes over time. In other words, a delete has occurred if the TID-Set of (at least) one partition has been associated to (at least) two distinct sets of tuples. To avoid such actions, the content of each partition must not change during the whole protocol: it must be made *invariant.* We define the `Invariance` safety property independently from the data structure to be made invariant.

**Definition 6 (`Invariance` safety property)** Let $L$ be a set of character strings containing the labels designating the data structures to be made invariant; $L$ is known by both the recipient and the secure devices. Let $l_0 \in L$ be a label, $b_0 \in \{0,1\}^*$ be an arbitrary bitstring, and $\texttt{Pr}[(l_0, b_0)]$ denote the probability that at least one secure device receives the couple $(l_0, b_0)$. We say that $(l_0, b_0)$ respects the `Invariance` property if for all bitstrings $b_i \in \{0,1\}^*$ received by any secure device, then $\texttt{Pr}[(l_0, b_i)] = 1$ if $b_i = b_0$, and $\texttt{Pr}[(l_0, b_i)] = 0$ otherwise.

For example, the set of tuples to be sanitized is invariant if the couple ("$\mathcal{T}^s$", $\mathcal{T}^s$) respects the `Invariance` safety property, "$\mathcal{T}^s$" being its label and $\mathcal{T}^s$ its actual bitstring representation.

*Implementations Sketches* The implementations of the `Origin` and `Identifier Unicity` safety properties are straightforward: when receiving a partition to sanitize, the given secure device simply checks the signatures of $s$-tuples and the absence of duplicate identifier[23].

The other properties are harder to check because they concern the complete dataset - we base their implementation on a *summary* of the dataset.

First, the summary contains for each partition a cryptographic hash of its $s$-tuples. Making the set of tuples to be sanitized become invariant consists in(1) letting the recipient send the summary to the absolute majority of a designated population of secure devices (e.g., the population might be the top 1% most available secure devices) - there can thus

[23] In general, the identifier can be implemented simply by letting secure devices generate a random number. It has to be big enough with respect to the number of tuples to collect in order to make collisions improbable so that in the rare collision cases the recipient simply keeps one of the colliding tuples. For example, around 5 billion numbers have to be generated to reach 50% chance collision with a 64-bits random number.

exist only one summary, and(2) when sanitizing a partition, letting the given secure device check that the summary was indeed sent to the required number of secure devices and that the actual hash of its $s$-tuples is consistent with the hash announced in the summary. The content of each partition thus satisfies `Invariance`.

Second, add to the summary the TID-Sets, expressed as ranges (for each partition, the min and max TIDs) and ordered (either ascending or descending). A linear scan of the summary is now sufficient for a secure device to assert that the summary meets `Mutual Exclusion`. Since it is invariant and its consistency is checked when sanitizing partitions, the recipient is forced to produce a single, mutually exclusive, partitioning - `Mutual Exclusion` is guaranteed. The `Membership` property is now trivial to check: when sanitizing a partition, the given secure device checks that the actual TIDs indeed belong to the TID-Set announced.

## A.2 Execution Safety of the Construction Function

Most global publishing models (if not all) share a common feature: their privacy guarantees are based on *a set of (one or more) counts*[24]. For example, the $(d, \gamma)$-Privacy guarantees depend on the number of true and fake records in the final release, $k$-Anonymity's on the number of records per equivalence class, and $\ell$-Diversity's on the distribution of sensitive values in each class. This recurrent need motivates the design of a secure counting protocol between the secure devices and the recipient; we call it `Secure Count`. `Secure Count` is a generic algorithm; when used to check the safety of the construction function's execution, `Secure Count` inputs the set of tuples to be sanitized $\mathcal{T}^s$ and outputs the correct corresponding set of counts (we sketch below a possible implementation). Checking these counts means asserting that `G(Secure Count(`$\mathcal{T}^s$`))` $=$ `True` where `G` is an algorithm-dependent check based on an algebraic manipulation of the counts. For example, the number of records in each class is counted and compared to $k$ for Mondrian, the distinct sensitive data within each class are counted and the compliance of their distribution with the chosen $\ell$-Diversity criterion is checked for Bucketization, and the numbers of fake and true records are counted and the compliance of their proportion is checked with respect to the expected $\alpha$ and $\beta$ for the $\alpha\beta$-Algorithm. Note that such counts are not private; they were already known to the recipient to build sanitization information respecting the privacy parameters of the algorithm.

Although counting tuples is a powerful primitive, it is not sufficient for all the possible algorithms: there exist algorithms whose sanitization information *is not fully decided by the recipient.* While this is not the case for the equivalence classes of our Mondrian and Bucketization instances, this occurs for the $\alpha\beta$-Algorithm: the true/fake nature of tuples in the $\alpha\beta$-Algorithm is set by secure devices during the collection phase and consequently must be checked (by devices too) for producing a sound count of true/fake tuples. For such algorithms, the unconstrained nature of their construction function precludes any generic checking mechanism. Fortunately, although full genericity cannot be reached here, the other safety properties defined above together with their implementations form a versatile *toolkit* that can be used for these algorithm-dependent checks. For example within the $\alpha\beta$-Algorithm, the algorithm-dependent checks consists in checking:(1) the authenticity of the true/fake nature of tuples (by signing also the true/fake bit in the `Origin` safety property),(2) the absence of any duplicate *record* within the set of tuples to be sanitized (by setting the tuple identifier to be its record's deterministic `MAC`),(3) the absence of any duplicate record outside the tuples to be sanitized (by splitting the allowed domains of collected true and fake tuples based on the `Invariance` safety property).As a result, we define below the `Safe Construction Function` safety property in charge of guaranteeing the execution safety of the construction function. The precise location of the enforcement of `Safe Construction Function` within the execution sequence will be given below in Section A.3. For the moment, it is sufficient to observe that the counts check must occur after having guaranteed the protection of the set of tuples to be sanitized (so that the `Secure Count` inputs a safe dataset), and before the final step of disclosing the sanitized records (so that

[24] Many common statistical computations can be built from a simple count primitive [9].

the recipient does not obtain sanitized tuples before the complete sequence of checks has been performed).

**Definition 7 (`Safe Construction Function` safety property)** The `Safe Construction Function` property is respected if both the counts check and the algorithm-dependent checks have succeeded.

*Implementing Secure Count* A naive count scheme could consist in using locks to synchronize secure devices on a shared counter: each participating secure device would lock the counter first, then increment it, and eventually sign it. However, due to the possibly high number of secure devices concurrently connecting, this approach would suffer from prohibitive blocking delays. In [2] we propose a count scheme free from locks. It consists in gathering on the recipient unitary *count increments* sent by secure devices, and in summing them up while ensuring the absence of tampering from the recipient. In the case where the set of counts overwhelms the resources of a secure device, [2] additionally proposes a scalable sum scheme inspired from traditional parallel tree-based sum schemes.

## A.3 Execution Sequence

As illustrated in Figure 9, the $\text{Met}_{\mathbb{A}}\text{P}_{(wm)}$'s sanitization phase consists in the following steps:

- **S1**: Apply `Invariance` to the partitioned set of tuples to be sanitized $\mathcal{T}^s$;
- **S2**: Assert `Mutual Exclusion` on the partitions of $\mathcal{T}^s$;
- **S3**: Process each partition (check the `Origin`, `Identifier Unicity`, and `Membership` safety properties, and return the encrypted sanitized records and the `Secure Counts`);
- **S4**: Check the counts for the `Safe Construction Information` safety property;
- **S5**: Apply `Unlinkability` to the set of encrypted sanitized records $\mathcal{T}^{v_o}$;
- **S6**: Finally decrypt the shuffled sanitized records to yield $\mathcal{T}^r$;

This execution sequence is composed of the generic steps of $\text{Met}_{\mathbb{A}}\text{P}_{(mal)}$ (where S1 and S2 can be performed in parallel, as well as S4 and S5). Specific instances whose sanitization information is fully decided by the recipient follow it as-is, the other instances additionally insert in it the algorithm-dependent checks of the `Safe Construction Function` property (see Section A.2). There only remains to protect its integrity (complete execution in the correct order) in order to cover all possible attacks (active and passive) from a recipient with weakly-malicious intent.

*Execution Sequence Integrity* Checking the completeness and order of the execution sequence is critical (e.g., final decryption must not occur before having checked the safety properties). The `Execution Sequence Integrity` safety property is respected if and only if the sequence flows through the *complete* sequence of safety properties in the *correct* order. Observe that an adversarial recipient essentially aims at obtaining records: he necessarily executes the first step - i.e., the initial $c$-tuples collection - and the last step - i.e., the final records disclosure. It consequently appears that completeness is guaranteed if the steps in between are all executed, and correctness if they are executed in the correct order. To this end, secure devices embed the *expected execution sequence* of the algorithm instantiated under $\text{Met}_{\mathbb{A}}\text{P}$, and control that the actual sequence matches the expected embedded sequence. Checking the match between the two sequences amounts to checking that, at each current execution step, the recipient is able *to prove* (in its most general meaning) that the *immediately preceding* step(s) was performed on the *expected data structure* (hence, by recursion, a valid *proof* for the current step demonstrates that the previous steps were *all* performed). Hence, when a secure device connects, the recipient sends to it the current execution step and the set of proofs binding the immediately preceding step to the data structure on which it was performed. The secure device checks the validity of the proofs, performs the task associated to the execution step, and finally returns the corresponding proof (in addition to the output of the task).

**Definition 8 (`Execution Sequence Integrity` safety property)** Let $S$ be the pre-installed set of execution steps, $s \in S$ denote the current execution step indicated to the connecting secure device by the recipient, $s.P$ denote the set of proofs required for executing $s$ as specified by $s.\texttt{VALID}$ (the detailed list of steps and proofs is given below). Finally, let $s.P_{rec}$ denote the set of proofs received from the recipient for step $s$. The `Execution Sequence Integrity` safety property is respected if: $\forall\, s \in S,$ then $s.\texttt{VALID}(s.P, s.P_{rec}) = \texttt{True}$.

The list of proofs required for each step of the generic MET$_{\mathbb{A}}$P$_{(mal)}$ execution sequence and their corresponding `VALID` function is the following:

- **S1** and **S2**: No proof required because no execution step precedes them.
- **S3**: The `Invariance` and `Mutual Exclusion` proofs are two signatures emitted by secure devices and binding for each property its label (e.g., "Invariance" and "Mutual Exclusion") to the hash of the summary to which the property was applied or checked. The set of proofs for S3 is thus the signatures plus the summary. The `VALID` function consists for each connecting secure device in verifying the signatures (correct labels and correct summary's hash) and checking the consistency of the summary with the partition to sanitize (TID-set and hash of its content).
- **S4**: The secure device that checks the counts asserts that all the partitions have been counted based on the partitions announced in the summary and on the partitions covered by the counts (secure counts include this information): this is S4's `VALID` function. The set of proofs for S4 consists of the counts and the summary (with its `Invariance` signature).
- **S5**: The set of encrypted sanitized tuples is first protected against tampering (the same way as $\mathcal{T}^s$ was). The set of proofs for the first shuffling level and the `VALID` function are thus the same as S3's but applied to $\mathcal{T}^{v_o}$. For the next shuffling levels, secure devices only need to check that the actual shuffling circuit is the expected one. The set of proofs is thus the expected shuffling circuit (e.g., computed from $\mathcal{T}^{v_o}$'s summary) and a signature binding each tuple to its current position in the circuit. The `VALID` function consists in verifying the tuples signatures and in checking that each set of tuples shuffled together is consistent with the shuffling circuit.
- **S6**: Finally, secure devices check the completeness of the shuffling circuit (i.e., the position of each tuple is at the end of the shuffling circuit), the signature obtained in S4 proving that the counts have been checked, and the continuity between $\mathcal{T}^s$ and $\mathcal{T}^{v_o}$ (based on the number of tuples they contain (e.g., computed from their summaries, or counted by the `Secure Count` scheme)): this is the `VALID` function. The set of proofs consists of the shuffling circuit and the positions of tuples (signed), S4's signature, and the number of tuples of $\mathcal{T}^s$ and $\mathcal{T}^{v_o}$ (e.g., computed from $\mathcal{T}^s$'s and $\mathcal{T}^{v_o}$'s summaries).

## B Extension to a Malicious$_{Hard}$ Recipient

### B.1 Typicality Property

The `Typicality` property is in charge of deterring the $Malicious_{Hard}$ recipient from performing attacks based on injecting forged tuples in the dataset. As explained in Section 6.3, the effectiveness of such attacks depends on the number of tuples forged by the recipient (based on the cryptographic keys of the cluster(s) it has broken) and injected in the dataset. The `Typicality` property consequently thwarts these attacks by requiring that the participations of all clusters be *similar*, where similarity can be instantiated in various ways (see below).

**Definition 9 (`Typicality`)** Let $\mathcal{P}$ denote a set of clusters participations (e.g., in the full dataset), and `T` denotes the statistical typicality test used by secure devices (e.g., a standard deviation analysis). $\mathcal{P}$ respects the `Typicality` safety property if $\texttt{T}(\mathcal{P}) = 1$.

Secure devices enforce the `Typicality` property by counting the participations of clusters (using the `Secure Count` protocol sketched above) and then checking that the set of counts satisfies the typicality test `T`. When the value of each count can be pre-defined (e.g., by fixing the number of tuples per cluster to be collected) `T` checks that each count is equal to the expected value. As a result, the $Malicious_{Hard}$ recipient is bounded by the size of the cluster on the number of tuples it can forge. This may be sufficient protection (e.g. for $\alpha\beta$-ALGORITHM), or it may still let the attacker cause severe privacy damage (e.g. for MONDRIAN). In any case, even if the participations cannot be predefined (e.g., within the $k$-ANONYMITY running example, the typicality can be defined inside each equivalence class), `T` is instantiated as a traditional statistical analysis, e.g., an outlier detection measure [8]. Section 7 shows the feasibility of the `Typicality` property both in terms of cost and detection effectiveness.

## B.2 Adapting the `Secure Count` and the Shuffle to clusters

We adapt the implementations of both the `Secure Count` and the shuffle following the same two-steps pattern. As a first step, the standard implementation is performed within each cluster. No modification is needed here. As a second step, the set of local - intra-cluster - results must be merged together in order to yield the global - inter-cluster - result: the counts obtained by all the clusters must be summed up, the tuples shuffled within each cluster must be shuffled with the tuples of all the other clusters. The difficulty that arises in the second step comes from the lack of trust a secure device can have concerning results originating from any cluster other than its own: it is unable to validate the signatures of the others, and is consequently unable to distinguish between data legitimately produced (e.g., the count of another cluster with its signature) and data injected by the recipient (e.g., a cheated count without any valid signature). We propose to deter such cheats by requiring that secure devices that participate in the second step simply *do not reveal to the recipient their cluster*: a cheat is detected if the connected device receives cheated data claimed to originate from its own cluster but with an invalid signature. At each connection, the probability of detecting the cheat is $1/n_c$, and the recipient needs to send the cheat to at least $n_c$ secure devices (to have each cluster compute its result). This worst case thus results in a lower bound detection probability of $1 - ((n_c - 1)/n_c)^{n_c}$. Note however that this worst case is unrealistic (e.g., in practice, in order to contact 500 clusters the recipient has to send the cheat to approximately 3400 secure devices). Let us now informally sketch the adaptation of the implementations of both the `Secure Count` and the shuffle.

The `Secure Count` takes as input the complete set of tuples and outputs the corresponding set of counts. We adapt it to secure devices following the two-step pattern described above: first, the count that is local to each cluster is computed as usual by the `Secure Count`, and second, the global count is computed by letting each cluster download the local counts and sum them. A malicious recipient may attempt to increase the global counts by sending forged counts, the rationale being to artificially reach the count required for the `Safe Construction Function` or the `Typicality` safety properties. However, since the secure devices do not reveal their cluster to the recipient, and because the recipient must obtain *all the local counts* because *each cluster will need the global count* later in the protocol, the high number of devices to which a cheated set of counts must be sent makes the detection probability reach highly deterring values as shown by Figure 14.

We follow a similar approach when adapting shuffling to clusters: first, shuffling that is local to each cluster is performed as usual, and second, global shuffling is performed on the outputs of local shufflings. The second step basically consists in letting each connected device (1) download a partition made of one tuple per cluster (previously shuffled locally), and (2) return the single tuple originating from its own cluster. During this step, as for the adaptation of `Secure Count`, the devices do not reveal their cluster to the recipient: the returned tuple is likely to originate from any cluster. A malicious recipient may cheat the second step by forming partitions that do not contain tuples from all the clusters (e.g., replacing some tuples chosen at random by a random bitstring): the shuffling would thus be incomplete. This cheat differs from the `Secure Count`'s cheat described above in that it is not necessary for the recipient to send a cheated partition to all the clusters: it can use a cheated partition to get one result tuple and then use the non-cheated version of the partition

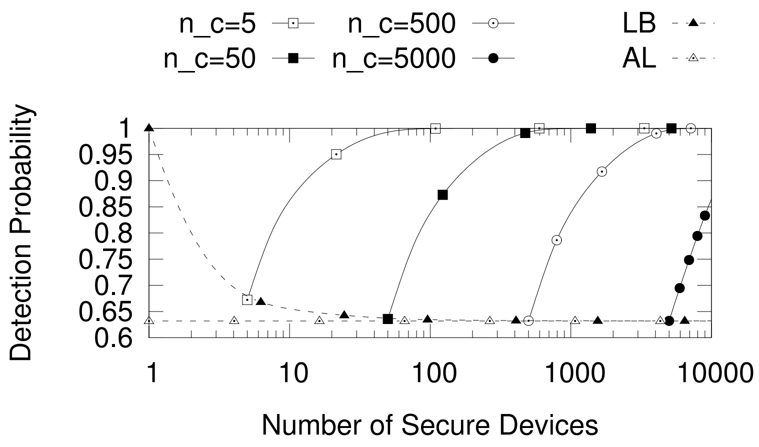


**Fig. 14** Evolution of the Detection Probability

to get the other tuples. The first protection measure is thus to make the partitions, either cheated or not, invariant (for each cluster). The second protection measure lies in hiding the connected devices' clusters, similarly as for `Secure Count`. Indeed, in order to obtain the results corresponding to the valid tuples of a partition, the recipient will have to send the latter to all the corresponding clusters. This results in a high number of devices to which a cheated partition must be sent, and consequently a detection probability that reaches sufficiently deterring values fastly.

Figure 14 shows the exponential growth of the detection probability with respect to the number of devices receiving a cheat (the x axis is at logscale). We plot the evolution of the detection probability for four numbers of clusters ($n_c = 5$, $n_c = 50$, $n_c = 500$, $n_c = 5000$) with respect to the number of secure devices that receive the cheat; their lower bound (LB) is also represented. We focus on the part of the curves above the asymptotic limit (AL) computed by :

$$\lim_{n_c \to +\infty} 1 - ((n_c - 1)/n_c)^{n_c} = 1 - 1/e$$

which is approximately 0.63.