TITLE

No Institute Given

Abstract. While most of the work done in Privacy-Preserving Data Publishing does the assumption of a trusted central publisher, this paper advocates a fully decentralized way of publishing anonymized datasets. It capitalizes on the emergence of more and more powerful and versatile Secure Portable Tokens raising new alternatives to manage and protect personal data. The proposed approach allows the delivery of sanitized datasets extracted from personal data hosted by a large population of Secure Portable Tokens. The central idea lies in distributing the trust among the data owners while deterring dishonest participants to cheat with the protocols. Deviant behaviors are deterred thanks to a combination of preventive and curative measures. Experimental results confirm the effectiveness of the solution.

1 Introduction

Privacy-Preserving Data Publishing (PPDP) attempts to deliver useful microdata sets for knowledge-based decision making without revealing the identity of individuals. A typical PPDP scenario starts by a collection phase where the data publisher (e.g., a hospital) collects data from data owners (e.g., patients). It is followed by a construction phase during which the sanitization rules are computed, and an anonymization phase where the publisher applies these rules to the data. The data can finally be released to a set of data recipients (e.g., a drug company or a public agency) for data mining or inquiry purposes.

Most research in the PPDP area considers a trusted model: the data publisher is trustworthy, therefore data owners easily consent providing it with their personal information [6]. This is unfortunately rarely the case in practice. There are many examples of privacy violations arising from negligence, abusive use, internal attacks, external attacks, and even the most secured servers are not spared[1]. ALL this has severely damaged individuals' trust in central servers. Moreover, legislation regulating the use of personal data usually authorizes statistical processing without the explicit consent of data owners, assuming this consent was given for the initial purpose of the data collection [1, 2]. A side effect is the reinforcement of individuals' suspicion towards the centralization of their data on any server. Their fears can even be high enough to halt nationwide projects [3] and many epidemiologists complain about the difficulty to get user's consent to participate in cohorts.

In this paper, we suggest a different way of publishing sanitized datasets with the objective to withdraw the major point of vulnerability introduced by

[1] http://datalossdb.org/

a central publisher. The core idea is to let each owner manage his data autonomously, under his own control, and participate in a distributed sanitization protocol using the storage and communication facilities provided by an untrusted publisher. This idea builds upon the emergence of new hardware devices called Secure Portable Tokens (SPTs for short). Whatever their form factor (SIM card, secure USB stick, wireless secure dongle), SPTs provide an unprecedented combination of portability (they are connectable to any terminal), secure processing (thanks to a tamper resistant smart card microcontroller) and mass stable storage (Gigabytes-sized NAND Flash chip). The use of SPTs for e-governance (citizen card, driving license, passport, social security, transportation, education, etc) is already actively investigated by many countries, and personal healthcare folders embedded in SPTs receive a growing interest, e.g., the Health eCard [2] in UK, the eGK card [3] in Germany, the HealthSmart Network [4] in the USA. Many kinds of personal data could actually be managed and protected thanks to a SPT with security guarantees stronger than those provided by any central server. This raises a natural question "How can we sanitize personal data embedded in Secure Portable Tokens without reintroducing a leak in the architecture?".

Answering this question means designing a protocol which produces an anonymized version of a database horizontally split among a population of trusted SPTs, such that the untrusted environment surrounding the SPTs can never learn more than the final result. This concern has been partially addressed by a limited number of works so far, in a way which unfortunately severely limits their practical scope. Secure Multi-party Computation protocols (SMC) allows several parties to jointly compute a function without revealing their input to one another [15]. Theoretically, any problem representable as a circuit can be securely solved, but the computation cost grows exponentially with the input size [8]. This disqualifies them for sanitizing widely distributed datasets. More efficient SMC constructs have been proposed to implement specific distributed PPDP protocols [17, 18, 10]. However, strong assumptions are made concerning the attack model (e.g., introduction of a Trusted Third Party in [10], absence of collusion between the Publisher and a Helper Third Party in [17]) and on the communication model, the underlying cryptographic protocols requiring broadcasting messages among all parties.

Does it mean that a general solution is unreachable? The problem is toughened by the SMC severe assumption that no one trusts anyone. The approach promoted in this paper actually makes the opposite assumption. It distributes trust among all data owners while deterring dishonest data owners to cheat with the protocols. Two ways of deterring deviant behaviors of parties are combined. The first way is preventive and relies on the tamper-resistance of each data owner's SPT. Although ultimate security does not exist, tamper-resistant hardware significantly increases the Cost/Benefit ratio of an attack. We have previously shown in a technical report (reference not included for the double

[2] `http://www.healthecard.co.uk`
[3] `http://www.gematik.de`
[4] `http://www.healthsmartnetwork.com/`

blind review process) that simple and secure PPDP protocols can be devised under the assumption that SPTs cannot be broken by any attacker. The second way is curative and relies on a mechanism detecting cheating parties, that is to say an attacker compromizing one or more SPT. This paper focuses on this second aspect.

Hence, this paper makes the following contributions. First, it proposes a probabilistic approach to detect compromized participants and show the effectiveness of this approach (the detection probability is close to 1). Second, it builds on this result to propose a new distributed PPDP computing model combining preventive and curative security measures. While the level of security reached by a probabilistic approach cannot be compared to SMC, we argue that it is high enough to meet the requirements of a broad set of applications, and notably Privacy-Preserving Data Publishing. Combining curative and preventive security significantly enlarges the scope and applicability of the approach. Our expectation is also that such a combination could pave the way for new solutions adapted to various privacy-preserving computing problems.

The rest of this paper is organized as follows. Section 2 details the problem statement by introducing a motivating scenario, stating the assumptions made on the distributed architecture and on the anonymization technique and fixing the attack model considered in this work. Section 3 briefly recalls the preventive way of deterring attacks. Section 4 introduces the curative way of deterring attacks and Section 5 evaluates its effectiveness. Finally, Section 6 concludes.

2 Problem Statement

2.1 Motivating example

The motivating example presented below capitalizes on a real experiment conducted in the field to improve the coordination of medical and social care for elderly people, while giving the control back to the patient over how her data is accessed and shared[5]. The ageing of population makes the organization of home care a crucial issue and requires sharing medical and social information between different participants (doctors, nurses, social workers, home helpers and family circle). Server-based Electronic Health Record solutions are inadequate because (1) the access to the folder is conditioned by the existence of a high speed and secure internet connection at any place and any time; and (2) they fail in providing ultimate security guarantees to the patients, a fundamental concern for patients facing complex human situations (diagnosis of terminal illness, addictions, financial difficulties, etc). This experimental project addresses these concerns as follows. Each patient is equipped with a SPT embedding a personal server managing her medical-social folder. As pictured in Figure 1, the form factor of patient's SPT is a USB token. A central server achieves the data durability by maintaining an encrypted archive of each patient's folder. The patient's

[5] The name and location of this experimental project are hidden for double blind review concern.

folder includes social information such as financial resources or scores measuring possible lack of autonomy, as well as medical data like diagnosis, treatments, and evolution of medical metrics (e.g., weight, blood pressure, cholesterol, etc.). This mix of medical and social information is of utmost interest for statistical studies.

When a practitioner visits a patient, the patient is free to provide her SPT or not, depending on her willingness to let the practitioner physically access it. In the positive case, the practitioner plugs the patient's SPT into his terminal, authenticates to the SPT server and can query and update the patient's folder according to his access rights, through a simple web browser. The patient's data never appears on any central server and no trace of interaction is ever stored in any terminal. If the patient looses her SPT, the SPT tamper-resistance renders potential attacks harmless. The folder can then be recovered from the encrypted archive maintained by the central server.

Now, let us assume that a health agency decides to collect sensitive data to perform an epidemiological study. Even paranoid patients will be disposed to consent participating in the study because: (1) they have the guarantee that their data will never be exposed on any server before being accurately anonymized, (2) they trust other patient's SPT to obey the protocol, knowing that tampering a SPT even by its owner is very difficult and (3) even in the improbable situation where a SPT is cracked, the cheater will be detected with a probability close to 1. When they are receiving care, their SPT does not remain idle but receives anonymization tasks from the agency and performs them in the background. Data used by these tasks is protected from prying eyes because it is kept confined in the SPT's secure environment. So, patients can enjoy their healthcare folder with full confidence without compromising neither their own rights to privacy nor any collective healthcare benefits.

Let us stress that the challenge tackled by this paper is not restricted to the healthcare domain. More generally, similar scenarios can be envisioned each time the legislation recognizes the right of the record owner to control under which conditions her personal data is stored.

2.2 Functional Architecture

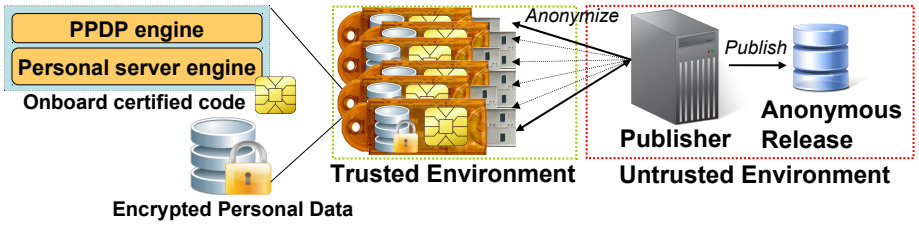


Fig. 1. Anonymous release of data stored on SPTs

Global Picture Figure 1 illustrates the functional architecture and modus operandi considered in the paper. The architecture is composed of two parts. The *Trusted Environment (TE)* is constituted by the set of SPTs participating in the infrastructure. Each SPT hosts the personal data of a single record owner. However, it can take part in a distributed computation involving data issued from multiple record owners since all the SPTs *a priori* trust each other. The number of participating SPTs is application dependent and may vary from tens of thousands in a small environment (e.g., a specific clinical study over a selected cohort) to millions in a region-wide or nation-wide initiative (e.g., an epidemiological study for a nation-wide health research program). The *Untrusted Environment (UE)* encompasses the rest of the computing infrastructure, in particular the data publisher and the data recipients. In the following, we make the simplifying assumption that UE has unlimited computing power and storage capacity, and is available 24/7.

The Secure Portable Token Regardless of their form factor, SPTs share several hardware commonalities. Their microcontroller is typically equipped today with a 32 bit RISC processor (clocked at about 50 MHz), ROM, small static RAM, a small internal stable storage (NOR Flash or EEPROM) and security modules providing tamper-resistance. The microcontroller is connected by a bus to a large external mass storage (Gigabytes of NAND Flash). Contrary to the microcontroller, this external mass storage is not hardware protected; hence data stored there must be encrypted, but the cryptographic keys and the encryption process remain confined within the microcontroller. SPTs can communicate with the outside world through various standards (e.g., USB2.0, bluetooth, 802.11) [11]. In summary, a SPT can be seen as a basic but very cheap (already today only a few dollars), highly portable, highly secure computer with reasonable storage and computing capacity for personal use. For illustration purposes, Fig. 1 depicts the SPT used in the experimental project described above and in our experiments.

The trustworthiness of SPTs lies in two factors: (1) the embedded software inherits the tamper resistance of the microcontroller making hardware and side-channel attacks highly difficult, (2) this software is itself certified according to the Common Criteria [6], making software attacks also highly difficult. This strongly increases the $\frac{Cost}{Benefit}$ ratio of an attack compared to a traditional server, considering also that a successful attack compromises only the data of a single individual. Note finally that the SPT owner herself cannot directly access the data stored locally; she must authenticate, thanks to a PIN code or a certificate, and only gets data according to her own privileges.

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

2.3 Privacy Model

The core idea of the approach proposed in this paper is independent of any privacy model. However, in order to favor a firm understanding we use the illustrative and popular k-anonymity privacy model [14] presented below.

We model the dataset to be anonymized as a single table $T(ID, QID, SD)$ where each tuple represents the information related to an individual hosted by a given SPT. ID is a set of attributes uniquely identifying an individual (e.g., a social security number). QID is a set of attributes, called *quasi-identifiers*, that could potentially identify an individual depending on the data distribution (e.g., a combination of Birthdate, Sex and Zipcode). The SD attributes contain sensitive data, such as an illness in the case of medical records. The table schema, and more precisely the composition of QID and SD, is application dependent. It is assumed to be defined before the collection phase starts, and is shared by UE and all SPTs participating in the same application (e.g., the same healthcare network).

The first anonymization action is to drop ID attributes. However, QID attributes can be used to join different data sources in order to link back an individual to its specific sensitive data with high probability. k-anonymity proposes to make such linkages ambiguous by hiding individuals into a crowd: it is often achieved by generalizing the QIDs to form equivalence classes (see [6] for a good overview), where each class contains (at least) k tuples sharing the same generalized QID'. Generalization-based algorithms (e.g., [13, 9]) partition the tuples into *equivalence classes* based on their QID values such that each class contains at least k tuples. An equivalence class is defined by a set of *generalization nodes* that specify the generalization of the QID values of the tuples belonging to the class. Figure 2 shows an example of raw data, a possible partitioning into equivalence classes containing $k \geq 2$ tuples, and the resulting 2-anonymous dataset. Anonymizing the tuples whose QIDs are in the equivalence class EC_1 simply means replacing their QID values by EC_1's generalization nodes, i.e., $\langle[75001, 75002], [22, 31]\rangle$.

2.4 Attack Model

Intents As stated in the introduction, most research in the PPDP area considers a trusted model. We could go one step further and consider the well-known

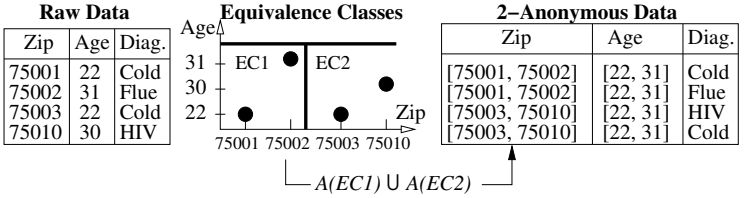


Fig. 2. 2-anonymous Equivalence Classes

Honest-but-Curious adversary model. In this model, an attacker obeys the protocol it is participating in but tries to infer confidential data by any indirect way. In this paper, we devise solutions acceptable by users who directly question the honesty of servers, either because they delegate part of their work to - potentially untrusted - subcontractors or because they are themselves vulnerable to internal and external attacks. So, we consider in this study a stronger adversary model called *Weakly-Malicious* [16]. In this model, an attacker cheats the protocol he is involved in only if (1) he is not detected as an adversary and (2) the final result is correct.

The weakly-malicious attack model fits particularly well the PPDP context. First, the longer an attack remains undetected, the bigger the benefit for the attacker. Second, the detection of an attack puts the attacker in an awkward position. This is true in all practical PPDP situations, whoever the attacker: (1) an insider within the PPDP organization, (2) the PPDP organization itself or (3) a subcontractor implementing the PPDP protocol on the organization behalf. Indeed, 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/public health damage (the halt of the Dutch EHR is an illustrative example) and they can even engage a class action.

Abilities The use of secure hardware in our solution leads us to slightly enrich the weakly-malicious attack model depending on the ability of the attacker to break one or more SPTs or not. Although breaking one SPT requires significant resources due to their proven high tamper-resistance, the attack could be launched if its benefits outweigh its cost. Hence we distinguish two variants of the weakly-malicious attack model:

- *Weakly-Malicious*$_{Soft}$: the attacker has *weakly-malicious* intent and the abilities of the attacker are said *Soft* in that it is unable to breach the hardware security of any SPT.
- *Weakly-Malicious*$_{Hard}$: the attacker has *weakly-malicious* intent and the abilities of the attacker are said *Hard* because it is able to break *at least one* SPTs and disclose its internal cryptographic material.

3 Weakly-Malicious$_{Soft}$ UE

This section overviews the Weakly-Malicious$_{Soft}$ protocol as a foundation for the Weakly-Malicious$_{Hard}$ protocol. We refer the interested reader to a technical report (withdrawn for double blind review concerns) for a more detailed discussion.

3.1 Rethinking the Traditional PPDP Phases

In the traditional (trusted) PPDP context, the publisher collects raw tuples of the form $\langle QID, SD \rangle$ during the Collection phase, computes equivalence classes

during the Construction phase, and replaces the QID of each tuple by the generalization node of the class it belongs to during the Anonymization phase. To defeat weakly-malicious attacks, the link between each QID and its related SD must be kept hidden throughout all the phases, while still allowing the publisher to compute and release k-anonymous classes. This leads to adapt the three phases of the protocol as follows.

During the **Collection phase**, each connecting SPT (that agrees to participate in the study) sends the publisher its QID in clear and its SD encrypted by a symmetric encryption scheme (e.g., based on the AES encryption function). The encryption scheme takes as parameter a secret key shared by all SPTs (key management is discussed in the next paragraph). The publisher decides to stop collecting tuples and to launch the **Construction phase** when it has received enough QIDs to build equivalence classes precise enough for its application-dependent requirements. During the **Anonymization phase**, any SPT that connects downloads a class (or more if its connection duration allows), and returns to UE the decrypted SDs it contains (in random order). The returned SDs are k-anonymous with certainty because the partial states observed by UE give no information allowing it to infer the association between a given SD and QID in clear with more precision than k.

Sharing cryptographic material among all SPTs does not hurt the Weakly-Malicious$_{Soft}$ assumptions since SPTs are considered unbreakable. We will relax this assumption in Section 4 when considering the Weakly-Malicious$_{Hard}$ attack model. We do the simplifying assumption that the SPT provider pre-installs the cryptographic materials inside the SPT's secured chip, though more dynamic protocols could easily be devised. Let us stress that even the SPT's owner cannot spy the hidden content or the computation made by her own SPT (in the same way as a banking card owner cannot gain access to the encryption keys pre-installed in her smart card microcontroller).

3.2 Weakly-Malicious$_{Soft}$ Actions

A Weakly-Malicious$_{Soft}$ UE can tamper the collected tuples based on either *Create*, *Destroy*, or *Copy* actions. Creating or destroying tuple(s) inside a given equivalence class directly leads to a reduction of the k-anonymity property (e.g., creating t fake tuples and adding them to $(k-t)$ collected tuples results in a $(k-t)$-anonymous class). Copying (i.e., duplicating) a set of tuples within a class (i.e., intra-class copy) also leads to a similar reduction of the k-anonymity property. Finally, copying a subset of tuples from a class C1 to another class C2 (i.e., inter-class copy) leads to inferences based on computing the differences between the respective SDs and QIDs of these two classes. Indeed, (1) the common SDs returned for both classes correspond to the common QIDs of these classes (i.e. the copied subset of tuples), and (2) the SDs returned for only one class correspond to the QIDs belonging to that class only. Thus computing the set-difference allows the UE to link subsets of collected QIDs to subsets of returned SDs with a probability less than $1/k$. These attacks are called *differential attacks*.

3.3 Safety Properties

Complete Coverage Safety properties aim at preventing the above actions. The *Origin* property guarantees that tuples originate from SPTs thereby preventing Create actions. By asserting that a class contains at least k tuples, the *Cardinality* property prevents Destroy actions. The *Distinguishability* property prevents intra-class Copy actions by asserting that all tuples are unique within a class. Finally, the conjunction of the following properties prevents inter-class Copy actions: the *Specialization* property requires each tuple to appear in a class that generalizes its QID (in other words, each tuple is at a legitimate place), the *Mutual Exclusion* property guarantees that the generalization nodes of different classes do not overlap (there is a single legitimate place for each tuple), and the *Invariance* property asserts that each class boundaries (i.e., its generalisation nodes) is always associated to the same tuples (the content of a class does not change over time).

Safety properties defeat the complete list of attacks UE can launch. It follows that processing equivalence classes satisfying all these properties will generate a k-anonymous result set with certainty.

Enforcement The *Origin*, *Cardinality*, *Distinguishability*, and *Specialization* properties can be easily checked during the Anonymization phase by a SPT on each downloaded class independently from the other classes. For example, enforcing *Origin* consists in checking for each tuple a message authentication code (MAC) [7] produced during the Collection Phase.

The *Mutual Exclusion* and *Invariance* properties concern the relation of each class to the whole set of classes already sent to SPTs. Checking them *would* require that SPTs share global information about the classes that each of them receives. There is no ultimate solution to this problem since (1) SPTs cannot communicate directly with one another and (2) cannot share a global history through UE, since UE can delete such information. Since each SPT relies on its own local history, UE could select the equivalence class sent to it such that all the properties are satisfied from the SPT viewpoint while they are violated from a global viewpoint. We deter UE from violating global properties by making any violation visible to SPTs through two caveat actions. The first caveat action precludes UE to control which SPT receives which equivalence class by using an anonymous communication channel [12] between UE and SPTs. The probability to send classes violating the global properties to the same SPT is no longer null. The second caveat action is to force UE to produce a *Summary* of the equivalence classes. This summary contains for all classes their generalization nodes plus a digest of their content (e.g., a hash of their tuples). Any SPT connecting during the Anonymization phase first downloads this summary to check the *Mutual Exclusion* and *Invariance* properties, and then downloads a class and checks its consistency with the summary. Hence, any tampering of a class must be cascaded to the summary, which is sent to all the connecting SPTs. Any SPT receiving two disagreeing summaries detects the attack. As a result, the detection is probabilistic, and its probability depends on the number

of SPTs receiving each version of the summary. The detection probability can be brought to highly deterring values (e.g., over 0.99) by tuning the minimal number of SPTs receiving each class. This can be done by setting this number to a low value (typically under 10).

3.4 Feasibility of the solution

For the sake of completeness, we summarize here the key results of our performance evaluation of the weakly-malicious$_{Soft}$ protocol. This protocol has been implemented and is being integrated in the architecture described in Section 2.1. The SPT hardware platform used in this architecture is still under test so the performance measurements have been conducted on a cycle-accurate hardware emulator provided by **(Remove for double blind?)** Gemalto (the world leader in smart cards), industrial partner of the project.

The measures have been performed with a synthetic population of 10^6 individuals, k varying from 10 to 100. The computing and transfer cost internal to each SPT amounted to couples of seconds, imputable mainly to transferring the summary and checking the global properties. The worst case occurred when k was minimal (8.8 seconds to treat a class when $k = 10$ while the cost is only 1.7 seconds when $k = 100$) because the number of classes was then maximal (and so was the size of the summary). These results confirm the feasibility of the solution, showing that it can match the performance requirements of many practical situations even in worst cases. Note that computing and transfer times scale linearly with the size of the population under study because of the linear increase of the summary's size. We also measured the latencies of the Collection and Anonymization phases. The main insight is that the latency of the Anonymization phase is insignificant with respect to the latency of the Collection phase. The interested reader will be referred to a technical report (withdrawn now for double-blind review concerns).

4 Weakly-Malicious$_{Hard}$ UE

In the previous Weakly-Malicious$_{Soft}$ protocol, if UE succeeds in breaking at least one SPT, it unveils not only the SPT's tuple but also its cryptographic materials which can in turn be used to decrypt the contents of all equivalence classes. To limit the scope of such attacks, the traditional solution is to use n different keys and organize the encryption process so that the impact of compromising one key is divided by n. Consequently, we partition SPTs into a set of *clusters*, denoted $\mathcal{C}$, *randomly* and *evenly*, such that SPTs belonging to different clusters are equipped with distinct cryptographic materials. Therefore breaking a SPT amounts to breaking a single cluster, and not the complete system anymore. However, it gives to the attacker the ability not only to decrypt data sent by SPTs that are members of the broken cluster, but also to encrypt data that originates from the broken cluster: weakly-malicious$_{Soft}$ Create actions can now pass the *Origin* property test.

This section first describes an adaptation of the Weakly-Malicious$_{Soft}$ protocol to a clustered world. Then it introduces a new safety property guaranteeing that weakly-malicious Create actions are harmless, and describes its enforcement. Finally, it describes the global Weakly-Malicious$_{Hard}$ protocol.

4.1 Clustered SPTs

Clustering cryptographic materials limits the decryption ability of SPTs to tuples originating from their own clusters. To tackle this limitation, SPTs participating in the Collection phase append the identifier of their cluster ID (CID) to the tuples sent to UE. Hence, each SPT participating in the Anonymization phase can ask to UE to send it a class into which its CID appears. However, a side effect of communicating to UE the CID of the connecting SPT is to reveal the CID of the returned anonymized tuples. UE would thus be able to link the returned tuples to the subgroup of collected tuples having the same CID. Since the subgroup's cardinality is most likely less than k, the returned tuples would be (less-than-k)-anonymous. To avoid this linking attack, the choice of the downloaded class must be made in stand alone by the connecting SPT, to which UE has previously sent the list of CIDs appearing in every classes.

Special care must also be taken of the number of anonymized tuples returned by a SPT. Indeed, a similar inference can be made by comparing this number to the cardinality of each subgroup of collected tuples sharing the same CID in the concerned class. To avoid this inference, SPTs downloading a same class must equalize the number of tuples they return; whatever their cluster, they must return at most GCD tuples, GCD being the greatest common divisor of the cardinalities of the subgroups inside the given class. To allow counting the number of tuples per cluster, each collected tuple t embeds an encrypted tuple and cluster identifier produced by the owner's SPT.

Transferring the complete list of CIDs per class between UE and a SPT can incur a significant network overhead. This overhead can be reduced by representing the list of CIDs of each class by a bitmap such that for all CIDs appearing in the class, the bit at index CID is set to 1. Each list is thus made of $|\mathcal{C}|$ bits and there are $|EC|$ lists. The total overhead amounts to transferring $|\mathcal{C}| \times |EC|$ bits. At the rate of 8Mbps (i.e., the current effective throughput measured on the hardware platform shown in Fig. 1), this does not present any bottleneck. Finding a class into which the SPT's CID appears has a negligible cost since it consists in checking a single bit per class bitmap.

4.2 Defeating Create Attacks

Weakly-malicious Create actions reduce the effective k-anonymity of a class by mixing j collected tuples, $j < k$, and injecting $(k-j)$ fake tuples forged thanks to the cryptographic materials of the broken cluster (let us assume for the moment that a single cluster is broken). The tampered class can contain far more forged tuples than legitimate ones. This is both the strength of the attack (it reduces the effective k in the same proportion) and its weakness (it is easier to detect).

Indeed, since SPTs are randomly and evenly partitioned into clusters, UE should receive roughly the same number of tuples per cluster, scattered uniformly into the equivalence classes. Inside a class affected by a weakly-malicious Create action, all clusters participate roughly with the same number of tuples, except the broken one that participates more than the others. We define a new *Typicality* safety property based on this observation stating that the participation of each cluster within a given class must be typical with respect to the participation of all other clusters. The above discussion can be generalized to an arbitrary number of broken clusters. Obviously, the more clusters are broken, the less atypical they are. However, breaking the hardware security of a single SPT is already a rather difficult task, making a massive weakly-malicious attack unrealistic.

SPTs enforce the *Typicality* property at the reception of a class, by analyzing statistical properties of the participation of clusters within the class. Section 5 demonstrates experimentally the efficiency of the attack detection by a straightforward analysis of the standard deviation of the participations in the class. 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 [5]), the standard deviation analysis already reaches high detection rates despite its simplicity.

4.3 Weakly-Malicious$_{Hard}$ Protocol

Algorithm 1 gives the execution sequence of the weakly-malicious$_{Hard}$ protocol from the Collection phase to the Anonymization phase. If a check is not fulfilled, the SPT stops the execution and raises an alarm (e.g., to the destination of the SPT owner or a trusted third party). Due to lack of space, we do not detail the mechanisms that are also part of the weakly-malicious$_{Soft}$ protocol (most notably the safety properties checks and duplicate removal).

5 Detection Probability

We consider a population under study of $N = 10^6$ individuals, randomly partitioned in $|\mathcal{C}| = 5.10^2$ clusters. In our experiments, all clusters are of equal size, but comparable results would be achieved with a normal distribution of individuals in clusters. The anonymization algorithm that we implemented divided the dataset into $|EC| = 8.10^3$ classes of at least $k = 10^2$ tuples each. Increasing the size of the population yields similar results in terms of detection. Since the distribution of SPTs to clusters is random, the clusters participation in a given class follows a normal distribution. To test the typicality of a cluster $C_j \in \mathcal{C}$ participating in the class $EC_i \in EC$, we compute σ the standard deviation (excluding non-participating clusters). In the general case, where $|\mathcal{C}| \gg k$ and there are no fake tuples created by UE, σ 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 3(a) shows the evolution of σ function of the number of tuples forged by a UE having broken a single SPT (then a single cluster). For instance, if

Algorithm 1 Weakly-Malicious$_{Hard}$ Protocol

Require: An anonymous communication channel between the SPTs and UE, the k-anonymity level, s the size of the population under study, $\mathcal{C}$ the clusters, the secret and private/public keys of clusters $\forall C_j \in \mathcal{C}\ \langle \kappa_j, \pi_j^{e|d} \rangle$ with their respective encryption/decryption functions $\langle E^{e|d}, P \rangle$.

1: **Collection phase:** For $i = 1...s$, each SPT_i, that connects sends to UE its tuple whose schema is $\langle QID, CID, Enc, Sign \rangle$, and whose value is $\langle QID_i, CID_j, E^e_{\kappa_j}(TID_i, SD_i), P_{\pi^e_j}(\langle TID_i, QID_i, SD_i, C_j \rangle) \rangle$, where TID_i is SPT_i's identifier, QID_i and SD_i his personal data, and CID_j his cluster's CID.

2: **Construction phase:** UE computes the set of equivalence classes EC and the summary $\mathcal{S}$ that contains for each class EC_j a bitmap representing the participating clusters ($EC_j.\beta$).

3: **Anonymization phase:**

4: **repeat**

5: SPT_i connects to UE. Let $SPT_i.\beta$ denote the bitmap representation of his cluster's identifier CID_j.

6: Download the summary $\mathcal{S}$.

7: **if** $\nexists\ EC_l.\beta \in \mathcal{S}$ s.t. $(SPT_i.\beta\ \text{AND}_{binary}\ EC_l.\beta) \neq 0$ **then**

8: Disconnect.

9: **else**

10: Download the EC_l's tuples $EC_l.T$ and their signatures.

11: Let c denote the number of distinct clusters in EC_l.

12: **end if**

13: Check the validity of tuples's signatures:
$\forall t \in EC_l.T, P_{\pi^d_j}(t.S) = \langle t.TID, t.QID, E^d_{\kappa_j}(t.Enc).SD, E^d_{\kappa_j}(t.Enc).CID \rangle$.

14: Count the participation of each cluster C_m appearing in EC_l:
$\forall m = 1...c,\ p_m = COUNT(t \in EC_l.T \text{ s.t. } t.CID = C_m)$.

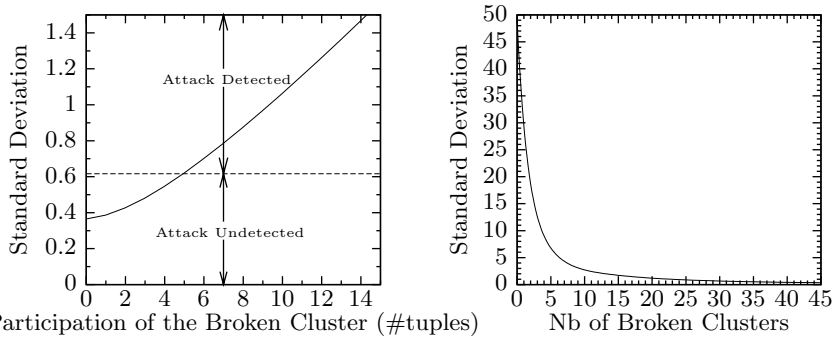
15: Check that the counts of participations are "typical".

16: Compute the greatest common divisor of clusters's participations:
$\gamma \leftarrow GCD(p_1, ..., p_c)$.

17: Send (at most) γ decrypted SDs from tuples that originate from CID_j.

18: **until** All classes have been completely anonymized

UE creates t tuples, then the class will contain only $k - t$ collected tuples. In order to achieve perfect knowledge of a target tuple (e.g., the tuple of a target individual identified by its QID), UE would need to inject $k - 1$ tuples (in our example, 99 tuples) in the class of the target tuple. As shown in Figure 3(a), a cluster participating more than 5 tuples leads to a statistically improbable value (i.e., $\sigma > \sigma_{max}$). Note that Figure 3(a) is a zoom: evolution is not linear but polynomial. If UE succeeds in breaking several clusters (meaning breaking SPTs from different clusters), fake tuples have less impact on the standard deviation because UE can distribute the participation over them. Figure 3(b) illustrates the value of σ function of the number of broken clusters b over which UE distributed evenly $k - 1 = 99$ created 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



(a) To the Participation of the Broken Cluster

(b) To the number of Broken Clusters

Fig. 3. Standard Deviation Sensitivity

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 weakly-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 SPTs and compromizing their hardware protections.

6 Concluding remarks

Version 0.1

In this article, we have presented a robust protocol to deal with the anonymisation of data produced by smart tokens, despite the fact that a small number of them could be broken. We illustrated this approach showing how the well-known Mondrian algorithm could be adapted, and secured against weakly-malicious attacks. More generally, attacks that can be devised will always be based on the same actions: Create, Destroy and Copy. Similarly, since deterrance resides in a probabilistic detection mechanism, the same sort of detection can be guaranteed for other types of algorithms.

The other specificity of our application is that it relies on secure hardware, which although cheap to produce is very costly to break. Yet the global approach is much broader. It could be adopted whenever a distributed application which is able to place *some* trust in participants wants to defend itself against weakly malicious attacks. Indeed, as we have shown in the experimentation, probabilistic detection is acheived even when a large number of clusters are broken, therefore

this means that our technique is applicable in contexts where the tokens are not "as secured" as the ones we use.

Version 0.2 (en cours)
Secure Multi-party Computation techniques fail in reaching a generic solution to Privacy-Preserving Data Publishing problems with an affordable cost. We believe that this is in part due to the strong "no-one-trusts-anyone" assumption. This paper promotes an approach based on the opposite assumption: all participants *a priori* trust each other. We propose to deter participants to adopt deviant behaviors (1) preventively by equiping participants with secure hardware and (2) curatively by detecting cheating parties.

We believe that this approach, illustrated above by the well-known k-anonymity privacy model, can be straightforwardly generalized to many distributed computation tasks for which participants are assumed to provide data in a uniform way. Actually, different applications may have different trust assumptions about participants. Some low-sensitive applications may not consider equiping participants with secure hardware and rely on the high detection probability to deter cheats, while more sensitive applications may need to rely on both to obtain a satisfying security level. The preventive and curative actions can indeed be used independently from the other to meet the desired security level.

Future work?

References

1. European Parliament and Council: Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data, 1995.
2. US Dept. of HHS: Standards for privacy of individually identifiable health information; final rule, 2002.
3. ICMCC. Dutch nationwide EHR postponed. Are they in good company?, 2009.
4. T. Allard, B. Nguyen, and P. Pucheral. Safe anonymization of data hosted in smart tokens. Technical Report 2010/25, PRISM Laboratory, 2010.
5. V. Barnett and T. Lewis. *Outliers in Statistical Data*. Wiley, 3rd edition, 1994.
6. B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey on recent developments. *ACM Comp. Surveys*, 2010. To appear.
7. O. Goldreich. *Foundations of Cryptography: Vol. 2, Basic Applications*. Cambridge University Press, 1st edition, 2004.
8. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the ACM Symp. on Theory of Computing*, 1987.
9. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *SIGMOD*, 2005.
10. F. Li, J. Ma, and J.-h. Li. Distributed anonymous data perturbation method for privacy-preserving data mining. *J. of Zhejiang University*, 10(7), 2009.
11. D. Praca. Next Generation Smart Card: New Features, New Architecture and System Integration. Deliverable of the Inspired IST project, 2005.
12. J. Ren and J. Wu. Survey on anonymous communications in computer networks. *Comput. Commun.*, 33:420–431, 2010.
13. P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 13(6), 2001.

14. L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5), 2002.
15. A. C. Yao. Protocols for secure computations. In *Proc. of the Symp. on Foundations of Computer Science*, 1982.
16. N. Zhang and W. Zhao. Distributed privacy preserving information sharing. In *VLDB*, 2005.
17. S. Zhong, Z. Yang, and T. Chen. k-anonymous data collection. *Inf. Sci.*, 179(17), 2009.
18. S. Zhong, Z. Yang, and R. N. Wright. Privacy-enhancing k-anonymization of customer data. In *PODS*, 2005.