

# Sanitizing Microdata Without Leak: Combining Preventive and Curative Actions

Tristan Allard, Benjamin Nguyen, and Philippe Pucheral

SMIS Project, INRIA Rocquencourt, 78153 Le Chesnay, France  
PRiSM Laboratory, 45, Av. des Etats-Unis, 78035 Versailles, France  
(First.Last)@inria.fr, (First.Last)@prism.uvsq.fr

**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 [3]. The advantage is that all the complex computing can be done on the central server. In this article, we get rid of the need of trusting central servers whose vulnerabilities to both negligences and internal/external attacks are frequently spotted<sup>1</sup>. This approach is possible thanks to the emergence of new hardware protected devices called Secure Portable Tokens (SPTs), on which a user can securely store personal data. This raises a natural question "How can we sanitize personal data embedded in Secure Portable Tokens without reintroducing a leak in the architecture?".

<sup>1</sup> <http://datalosssdb.org>



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. The generic Secure Multi-party Computation (SMC) construct allows several parties to jointly compute a function without revealing their input to one another [9], but its cost grows exponentially with the input size [4]. This disqualifies it for sanitizing widely distributed datasets. More efficient SMC constructs have been proposed to implement specific distributed PPDP protocols [11, 12, 6]. However, strong assumptions are made concerning the attack model.

The approach promoted in this paper makes a novel 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 the SPTs. We have previously shown in [1] 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, i.e. an attacker having broken 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 compromised participants and shows the effectiveness of this approach. 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.

The rest of this paper is organized as follows. Section 2 details the problem statement through a motivating scenario, illustrating the assumptions made on the architecture, anonymization technique, and considered attack model. 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

**Motivating example:** The motivating example presented below capitalizes on a real experiment conducted in the field called PlugDB<sup>2</sup>. PlugDB aims at improving 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. 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

---

<sup>2</sup> <http://www-smis.inria.fr/~DMSP/home.php>

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 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 loses 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.

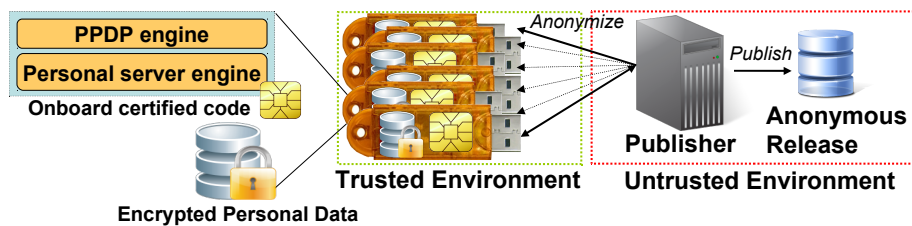


Fig. 1. Anonymous release of data stored on SPTs

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.

**Functional Architecture:** 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 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 <sup>3</sup>, 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. In summary, a SPT can be seen as a very cheap (a few dollars today), highly portable, highly secure computer with reasonable storage and computing capacity for personal use.

**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 [8].

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).

<sup>3</sup> <http://www.commoncriteriaportal.org/>

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 his own sensitive data with high probability. *k*-anonymity proposes to make such linkages ambiguous by hiding individuals into a crowd. It is often achieved through a mechanism called *generalization* (see [3] for a good overview). Generalization based algorithms (e.g., [7, 5]) partition tuples into *equivalence classes* containing at least *k* tuples with similar *QID*, and for each class release a single coarsened *QID'* together with the tuples's *SDs*. 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 the range  $\langle [75001, 75002], [22, 31] \rangle$ .

**Attack Model:** As stated in the introduction, most research in the PDP area considers a trusted model. We could go one step further and consider the well-known *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 these latter 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* [10]. 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 PDP 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 PDP situations, whoever the attacker: (1) an insider within the PDP organization, (2) the PDP organization itself or (3) a subcontractor implementing the PDP 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.

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

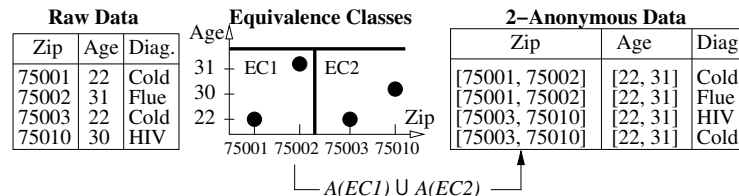


Fig. 2. 2-anonymous Equivalence Classes

if its benefits outweigh its cost. Hence we distinguish two variants of the weakly-malicious attack model:

- *Weakly-Malicious<sub>Soft</sub>*: 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<sub>Hard</sub>*: 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<sub>Soft</sub> UE

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 coarsens the  $QID$  of each tuple according to 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<sub>Soft</sub> assumptions since SPTs are considered unbreakable. We will relax this assumption in Section 4 when considering the Weakly-Malicious<sub>Hard</sub> 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).

We identified in [1] the Weakly-Malicious tampering actions over the dataset that lead to disclosures. To prevent any tampering, the trusted environment, i.e., the SPTs, is in charge of enforcing a (small) set of properties over the dataset. For space reasons we do not detail them but refer the interested reader to [1].

## 4 Weakly-Malicious<sub>Hard</sub> UE

In the previous Weakly-Malicious<sub>Soft</sub> 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. This is clearly undesirable because creating  $t$  fake tuples and adding them to  $(k - t)$  collected tuples will eventually result in a  $(k - t)$ -anonymous class.

**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 (*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 class.

Special care must also be taken about 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. The collected tuples already contain their originating *CID* so SPTs participating in the Anonymization can easily count the number of tuples per cluster. However, they must also check that each tuple is legitimately bound to the good *CID*. To this end, each SPT participating in the collection phase signs his tuple using his cluster's cryptographic material and sends to UE the tuple with its signature. Signatures are then checked by SPTs participating in the Anonymization phase.

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

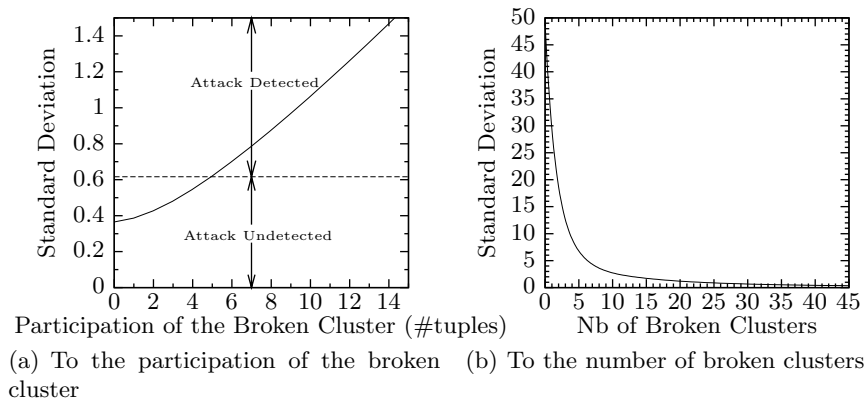
and there are  $|EC|$  lists. The total overhead amounts to transferring  $|\mathcal{C}| \times |EC|$  bits. At the rate of 8Mbps (i.e., the current effective throughput measured on the hardware platform shown in Fig. 1), this does not present any bottleneck. Finding a class into which the SPT's *CID* appears has a negligible cost since it consists in checking a single bit per class bitmap.

**Defeating Weakly-Malicious Creations:** Weakly-malicious tuples creations 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 infected by weakly-malicious created tuples, all clusters participate roughly with the same number of tuples, except the broken one that participates more than the others. We define the *Typicality* property based on this observation. The *Typicality* property states 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. The *Typicality* property can be straightforwardly enforced at the reception of a class, by analyzing statistical properties of the participation of clusters within the class.

## 5 Detection Probability

We consider a population under study of  $N = 10^6$  individuals, randomly partitioned in  $|\mathcal{C}| = 5 \times 10^2$  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  $|EC| = 8 \times 10^3$  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  $\sigma$  the standard deviation (excluding non-participating clusters). In the general case, where  $|\mathcal{C}| \geq k$  and there are no fake tuples created by UE,  $\sigma$  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 3(a) shows the evolution of  $\sigma$  function of the number of tuples forged by a UE having broken a single SPT (then a single cluster). For instance, if 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



**Fig. 3.** Standard Deviation Sensitivity

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  $\sigma$  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 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., traditional outlier detection measures [2], combining several statistical measures, choosing 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<sub>hard</sub> 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 compromising their hardware protections.

## 6 Concluding remarks

The increasing suspicion of individuals towards the centralization of their sensitive data on servers is established. This urges the scientific community to find credible distributed alternatives to the usual PPDP computing model based on a trusted central publisher. Secure Multi-party Computation techniques fail in

providing a generic solution with an affordable cost. This is partly 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. This approach deters participants to adopt deviant behaviors (1) preventively by equipping them with secure hardware and (2) curatively by detecting cheating parties. We have shown that the detection probability of attacks is close to 1 making the solution well adapted to a *Weakly-Malicious* adversary model, a model capturing well the security requirement of a PPDP framework.

Our future work is twofold. First, we must validate the adequacy and practical interest of the solution through the forthcoming experiment in the field sketched in this paper. Second, we believe that this approach, illustrated in this paper by the well-known *k*-anonymity privacy model, can be generalized to many other distributed computing problems. Indeed, the distinction introduced in this paper between *Weakly-Malicious<sub>Soft</sub>* and *Weakly-Malicious<sub>Hard</sub>* attacks characterizes a security model based on (1) *a priori* trust and (2) detection of cheating party, independently from the underlying protocol. The *Weakly-Malicious<sub>Hard</sub>* definition could be easily reformulated to embrace situations where *a priori* trust is broken with no explicit reference to any secure hardware device.

## References

1. T. Allard, B. Nguyen, and P. Pucheral: *k*-Anonymizing Data Hosted in Smart Tokens with a Weakly-Malicious Publisher. Technical report 2011/28, PRISM Laboratory (2011)
2. V. Barnett and T. Lewis: *Outliers in Statistical Data*. Wiley, 3rd edition, 1994
3. 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 (2010)
4. O. Goldreich, S. Micali, and A. Wigderson: How to Play any Mental Game. In: *ACM Symp. on Theory of Computing* (1987).
5. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan: Incognito: Efficient Full-Domain *k*-Anonymity. In: *SIGMOD* (2005)
6. F. Li, J. Ma, and J.-h. Li: Distributed Anonymous Data Perturbation Method for Privacy-Preserving Data Mining. *J. of Zhejiang University* 10, 952–963 (2009)
7. P. Samarati: Protecting Respondents’ Identities in Microdata Release. *IEEE TKDE* 13, 1010–1027 (2001)
8. L. Sweeney: *k*-Anonymity: a Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 10, 557–570 (2002)
9. A. C. Yao: Protocols for Secure Computations. In: *Symp. on Foundations of Computer Science* (1982)
10. N. Zhang and W. Zhao: Distributed Privacy Preserving Information Sharing. In: *VLDB* (2005)
11. S. Zhong, Z. Yang, and T. Chen: *k*-Anonymous Data Collection. *Inf. Sci.* 179, 2948–2963 (2009)
12. S. Zhong, Z. Yang, and R. N. Wright: Privacy-Enhancing *k*-Anonymization of Customer Data. In: *PODS* (2005)