UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Département d'informatique
UFR de Sciences

École doctorale "STV"

Sanitizing Microdata Without Leak
A Decentralized Approach

THÈSE

présentée et soutenue publiquement le lundi 12 décembre 2011

pour l'obtention du

Doctorat de l'université de Versailles Saint-Quentin

(spécialité informatique)

par

Tristan ALLARD

Composition du jury

Directeur :	Philippe PUCHERAL	Professeur, Université de Versailles Saint-Quentin.
Encadrant :	Benjamin NGUYEN	Maître de Conférences, Université de Versailles Saint-Quentin.
Rapporteurs :	Elisa BERTINO	Professeur, Purdue University.
	David GROSS-AMBLARD	Professeur, Université de Rennes.
Examinateurs :	Fosca GIANNOTTI	Directeur de Recherche, ISTI-CNR.
	Philippe AIGRAIN	Directeur, Sopinspace.
	Guillaume RASCHIA	Maître de Conférences, Université de Nantes.

Laboratoire PRISM

informatiques mathématiques Inria

Remerciements

Je remercie aussi chaleureusement que la bienséance me le permet Philippe et Benjamin. Collaborer avec vous a été source d'apprentissages quotidiens. Vos qualités de rigueur, de bon-sens, de bienveillance, d'intégrité, de conscience professionnelle, ont fait de ces quelques années passées ensemble une grande source d'inspiration.

Merci à Georges Gardarin pour m'avoir ouvert les portes de la thèse.

My grateful thanks go to Elisa Bertino, David Gross-Amblard, Fosca Giannotti, Philippe Aigrain, and Guillaume Raschia for having accepted to be part of my Ph.D. jury. It is a great honour for me to gather you around this thesis. I especially warmly thank the reviewers Elisa Bertino and David Gross-Amblard.

Merci aux membres de l'équipe SMIS, qui contribuent à faire de SMIS un environnement de travail stimulant et plein de joie et d'humour. Shaoyi, merci pour cette agréable cohabitation!

Enfin, j'adresse un très grand merci à tous ceux qui m'ont donné leur avis plein de bon sens sur un dessin, une phrase, un argument (Jo, Ghislain, Babeth, Erwan). Et tous ceux qui ont supporté indirectement cette thèse - le plus souvent inconsciemment - par leur attention, patience, ou générosité.

À l'heure du thé.

Contents

Chapter 1

Introduction

Summary: *The frontiers between humans and the digital world are tightening. An unprecedented and rapidly increasing amount of individual data ends up nowadays into well-structured and consistent databases. Privacy-preserving data publishing models and algorithms are an attempt for benefiting collectively from this wealth of data while still preserving the privacy of individuals. However, few works have focused on their implementation issues, leading to highly vulnerable practical scenarios. This thesis tackles precisely this issue, benefiting from the rise of portable, large capacity, and tamper-resistant devices used as personal data servers. Its contribution is a set of correct and secure protocols permitting to perform traditional non-interactive privacy-preserving data publishing algorithms and models in an original context where each individual manages her data autonomously through her own secure personal data server.*

Contents

1.1 Privacy at Stake

Computer technologies pervade our daily lives, seamlessly blurring the frontiers between the digital and non-digital worlds. *"Ever more pervasive, ever harder to perceive, computing has (...) insinuated itself into everyday life."* notes Adam Greenfield when introducing ubiquitous computing in [Greensfield06].

Almost every aspect of our life is now digitalized. Modern access control to public transportation is based on electronic passes and individuals are often equipped with geolocalization devices (e.g., car's navigation systems, smart-phones), shopping baskets are logged, just like credit cards transactions, communications often run through the Internet (e.g., electronic mails, phone calls, instant messaging), nations are launching ambitious electronic health record systems (EHR), information is accessed through the web (Google's servers processed over one billion search queries per day in 2009 [Kuhn09]), and so on. A recent report [Gantz11] estimates at more than 1.8 zettabytes[1] the total amount of digital information created or replicated in 2011, assigning to individuals the generation of 75% of this information (the major part of which being unconsciously generated - e.g., search logs).

In this digital-infused world, the ever-growing digital footprints of individuals usually end up in well-structured and consistent databases in charge of supporting efficiently their daily use.

Big Benefits (and Big Risks). This unprecedented abundance of individual data is a ground-breaking opportunity for human societies. For example, analyzing medical data could result in various benefits [Safran07, PWC09, Savage11] including improved patient care, better trend predictions of public health, and reduced healthcare costs. Allowing academics and industrials to access such a wealth of data is thus crucial, but it comes at a cost: the privacy of individuals is more than ever at risk. Aware of the possible benefits of *secondary data uses*[2], legislation is usually permissive provided that the data has been rendered *"anonymous"* [EPC95, HHS02]. But what does "anonymous data" mean in the Internet age?

Two striking cases illustrate the non-triviality of the answer. First, in 2002, Sweeney identified the medical record of the governor of Massachusetts in a medical dataset that was believed to be anonymous (names and social security numbers had been removed) and had been published for research purposes [Samarati98, Sweeney02]. The link between the medical record and the governor was restored by joining the medical dataset with the Cambridge Massachusetts's voter list on the zipcode, birth date,

[1] A zettabyte is equal to one billion terabytes, i.e., 10^{21} bytes.

[2] A *secondary use* refers to the use of data for a purpose different from the initial data collection's purpose (e.g., statistical analysis of health information stored in an electronic health record).

and gender attributes. Studies indeed show that a majority of US citizens is uniquely identifiable based on these attributes [Sweeney00, Golle06]. Second, in august 2006, AOL™published 20 Million search queries issued by 658 000 users for research purposes after having mapped each user-name to a random identifier [Arrington06]. Few days after, the New York Times disclosed the identity of user No. 4417749 based on the keywords contained in her search queries [Barbaro06], and AOL™removed hastily the dataset from its website.

These disclosures were made easy because of the use of simplistic *pseudonymization* schemes for sanitizing data. Loosely speaking, pseudonymization is the mere replacement of some attributes (typically, the attributes that are considered as directly identifying) by a pseudonym (e.g., a pseudo-random number). The rest of the attributes is let intact to favor utility. For example, in the medical dataset analyzed by Sweeney, the names and social security number were removed from the dataset, and the social and medical information were let intact (whereas they contained both identifying and sensitive data).

Privacy Versus Utility. *Not publishing* a dataset is a straightforward means to protect its privacy... but also to drastically reduce its utility. Conversely, publishing a raw dataset without *transforming* it yields full utility but completely thwarts its privacy.

These two ways of publishing a dataset are actually the extreme points of the non-interactive[3] *Privacy-Preserving Data Publishing* (PPDP for short) spectrum. Privacy models and algorithms allow the *data publisher* (e.g., a hospital) to place the cursor on the PPDP spectrum. A privacy model defines formally the meaning of *privacy*, and allows the publisher to specify a lower bound on the privacy guarantees offered by a published dataset (e.g., a table containing patients's health records). A privacy algorithm transforms the raw dataset into a dataset whose privacy is higher than the lower bound and which utility is (as near as possible) to optimal, where optimality is defined by an appropriate *utility metric*. Transforming a dataset containing raw personal data into a dataset that preserves privacy is called *data sanitization*. In practice, a dataset is potentially sanitized in different forms, e.g., based on different privacy models, different privacy lower bounds. Each resulting sanitized dataset is finally released to a *data recipient* (e.g., a drug company, a public agency).

Current Implementations: a Major Source of Vulnerabilities. Although a substantial amount of work has been done on privacy models and algorithms (see the surveys [Chen09a, Fung10]), trying to reach the best privacy/utility tradeoff, far

[3] We do not consider in this thesis the interactive approach to privacy-preserving data publishing which answers statistical queries posed over a private dataset rather than sanitizing and publishing the dataset (see Section 2.2).

less attention has been paid to how these models are actually implemented in practice. This raises two major issues.

First, most PPDP works consider that the data publisher is trustworthy, so that the complete PPDP process can be centralized [Chen09a]. This assumption is unfortunately rarely verified 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[4]. This severely damages individuals' confidence in central servers[5] and user's consent to participate in PPDP studies is usually very hard to get.

Second, to different recipients are intended different sanitized datasets (e.g., epidemiologists receive data of higher quality than the pharmaceutical industry). For practical reasons, this process is usually organized as follows[6]: (1) nominative data is extracted from an OLTP server (e.g., the central EHR), (2) this data is simply pseudonymized and stored in a warehouse, and (3) various sanitized datasets are computed on demand from this pseudonymized version (e.g., epidemiologists receive data of higher quality than the pharmaceutical industry). However, this process introduces a major source of vulnerability regarding the data at rest in the warehouse because (1) pseudonymization is recognized as a legal form of sanitization [Quantin08, Neubauer09] and (2) the legislation regulating the management of sanitized data is rather permissive (no retention limit, no encryption of stored data, no user's consent required).

1.2 A Decentralized, Generic, and General Approach

We address in this thesis the problem of implementing privacy-preserving data publishing models and algorithms with strong guarantees of security. To this end, we suggest a radically different way of producing sanitized datasets with three main objectives in mind:

- **Decentralization**: each individual manages her data autonomously, under her control, and participates voluntarily in a sanitization protocol[7]. Hence, we get rid of the trusted central publisher assumption.

[4] http://datalossdb.org/

[5] The Dutch national Electronic Health Record project was halted because of the numerous privacy concerns expressed by individuals [Bos09].

[6] The French and UK EHR system are built on this principle.

[7] For the writing clarity, we use the term *protocol* and not *distributed algorithm*; we stress that this thesis is not related to the formal study of communication protocols.

- **Genericity**: the protocol must support the production of datasets answering different privacy/utility trade-offs to be intended to different recipients. Hence, it must (as far as possible) be agnostic with respect to privacy models and algorithms.

- **Generality**: the protocol must adapt to realistic environments; it must be scalable up to nationwide sized datasets and must not rely on strong availability assumptions concerning the participants.

The proposed solution builds upon the emergence of new decentralized models to manage personal data, like the FreedomBox initiative[8] or the Personal Data Server approach [Allard10a]. Both rely on the use of a cheap and secure personal server which always remains under the holder's control. *Secure Portable Tokens* (or tokens for short) can be used to implement secure personal servers. Whatever their form factor (mass storage SIM card, secure USB stick, wireless secure dongle), tokens combine tamper-resistant micro-controllers (protected against illegitimate uses - including those of the owner) with Gigabytes of NAND Flash storage. Today, the use of tokens for e-governance (ID cards, driver's licenses, transportation passes, school IDs, etc) is actively investigated by many countries, and personal healthcare folders embedded in tokens receive a growing interest (see, e.g., the Health eCard[9] in UK, the eGK card[10] in Germany, the LifeMed card[11] in the USA).

Implementing a distributed PPDP protocol on this basis is rather trivial given that the sanitization algorithm is based on *local perturbation* [Agrawal09]: the protocol merely consists in letting each participant perturb his own data *independently* of each other (see Chapter 2 for an introduction to local perturbation models and algorithms). However, much better utility/privacy trade-offs can be reached by *centralized publishing* algorithms (as shown in [Rastogi07]) essentially because they sanitize each piece of data based on the knowledge of the complete dataset. The challenge tackled in this thesis is precisely to enable centralized publishing algorithms to be used while answering the **Decentralization**, **Genericity**, and **Generality** objectives. Note that we focus on a single-release context where the initial dataset is sanitized once for all. We will discuss in Section 7.3 pursuing this work in a context where the dataset evolves over time and is sanitized on a regular basis.

[8] http://www.freedomboxfoundation.org/
[9] http://www.healthecard.co.uk
[10] http://www.gematik.de
[11] http://www.lifemedid.com/

1.3 Contributions

The study conducted in this thesis took place chronologically in three steps. We started the protocol study by focusing on a specific privacy setting (i.e., restricted to *generalization-based* algorithms in the context of the k-Anonymity model), and requiring the protocol's execution to be as simple as possible. These assumptions simplified the problem and led us to the first contribution: the Protogen suite of protocols (meaning protocol for generalization) [Allard11c, Allard11a, Allard11b]. Then, dropping the simplifying assumptions, we dissociated the elements exclusive to the working of the protocol from those exclusive to the generalization-based algorithms, and abstracted them such that they be as far as possible unrelated to any specific type of algorithm. This resulted in the second contribution, namely the Meta-Protocol suite of protocols. At the time of writing this thesis, this work has not been submitted yet. In addition, we also designed, in a preliminary study, a practical architecture supporting an electronic health record system based on tokens [Allard09, Allard10c, Allard10b]. The third contribution of this thesis consists in this architecture, in which we root a discussion on the practical aspects of an implementation of the Protogen and Meta-Protocol suites of protocols. We describe below each of the thesis's contributions.

The Protogen and Meta-Protocol Suites. Supporting centralized publishing algorithms in a decentralized context requires sharing some algorithm-dependent information about the complete dataset. However, disconnected and autonomous tokens can hardly support this data sharing alone because of their low availability. They consequently rely on an *untrusted publisher* in charge of supporting the exchange of information, but potentially adopting adversarial behaviors. The Protogen and Meta-Protocol suites are designed on the same mold, considering attack models incrementally stronger.

First, a passive adversary model, called the *honest-but-curious* attack model, allows us to lay the foundations of each suite of protocols by designing clean execution sequences free from data leaks. The execution sequences are a remodeling of the traditional phases of centralized publishing algorithms. In a trusted publisher context, the publisher first *collects* raw individuals's records, then *constructs* some algorithm-dependent data based on the collected records (the resulting data structure is precisely the reason for which the algorithm is centralized), and finally *sanitizes* the records based on the data structure previously computed. In our context, unveiling the raw records to the untrusted publisher is a blatant privacy breach. We need to obfuscate them while still allowing the publisher to construct the algorithm-dependent data structure. Indeed, the fundamental principle of our approach is the delegation of the construction task to the publisher (made possible by the disclosure of a controlled, algorithm-dependent, amount of information about each tuple) while the rest of the

execution sequence remains under the responsibility of tokens. Through a careful design of the information exchanged and the operations performed during their respective execution, PROTOGEN is not tied to any given generalization-based algorithm, and META-PROTOCOL to any specific privacy model or algorithm.

Second, we upgrade the primary execution sequence to tackle a stronger, active, adversary that may tamper the protocol if it is sure that the attack will not be detected and the result remains correct; such attack model is called *weakly-malicious*. Based on the observation that any attack is an arbitrary combination of basic *tampering actions* (i.e., forge, copy, delete), we preclude completely the latter through the definition of a small set of *safety properties* that must be asserted by tokens during the execution sequence. We also propose a proof-of-concept implementation of each safety property which demonstrates their practical feasibility.

These two attack models especially fit real-life untrusted publishers: the passive one could correspond to a well-established publisher (e.g., a government agency), the active one could correspond to a more questionable third-party publisher (e.g., a software-as-a-service company).

The two first contributions of this thesis consist in the PROTOGEN and META-PROTOCOL suites. Their security and correctness are demonstrated and thorough experimental validations assess their feasibility. Note that the PROTOGEN and META-PROTOCOL suites are not based on complex cryptographic constructs, but on a simple use of traditional cryptographic properties (that are typically provided by well-known cryptographic schemes such as, e.g., the *Advanced Encryption Standard* (AES) and the *Secure Hash Algorithm* (SHA) - see Section 2.3). We designed both suites of protocols according to the two-stage process traditionally followed by cryptographic schemes proposals: a definition step (i.e., *what are the desirable properties?*) and a construction step (i.e., *how do we enforce them?*).

The Token-Based Electronic Health Record System. Many nations around the world have launched ambitious electronic health record projects. These systems are highly topical because of both their promises and challenges (notably concerning data privacy and availability). The approaches vary from systems fully centralized (e.g., the French DMP, the VistA system) to systems fully decentralized based on smart-cards (e.g., the UK's Health eCard, the Pittsburgh's Health Passport Project).

We propose an alternative architecture based on investigations conducted in the PlugDB project[12]. PlugDB aims at embedding a complete chain of software within the token's secure environment (i.e., operating system, database management system, web-server) to pave the way for the design of full-fledged secure personal data servers.

[12]PlugDB (http://www-smis.inria.fr/~DMSP) is a research project investigated by the SMIS team and funded by the French national research agency (see Chapter 6 for details).

Indeed, the token's storage capacity, orders of magnitude superior to the smart-cards's capacity used in the current EHR projects, allows it to embed (the most significant part of) its owner's medical folder. PlugDB offers the opportunity to discuss a possible real-life implementation of the protocols proposed in this thesis. Data availability is an issue here since tokens are not always connected and may also get lost or broken. We solve this issue thanks to a supporting central server and a simple *pedestrian* synchronization protocol. We consider two types of individual data uses: the daily medical use and the privacy-preserving data publishing use, and propose a data classification defining precisely how each piece of data should be handled, based on the individual's own wish for privacy. Finally, we discuss the practical aspects related to setting and executing the privacy-preserving data publishing protocols. The third contribution of this thesis consists thus in this EHR architecture, including the pedestrian synchronization protocol and the data classification, and the final informal discussion.

1.4 Illustrative Scenario

Alice is an eighty years old woman interested in technology. She is one of the early adopters of the new health record system based on secure portable tokens. When she visits (or is visited by) her physician, the latter accesses Alice's health record through her token. Alice is confident that the privacy of her personal data is protected thanks to the strong security guarantees provided by the token and by modern encryption schemes. The statistical agency of the country where Alice lives has recently launched a national epidemiological survey. Sanitized data will be released, in different forms, to both the statistical agency's epidemiologists and to researchers of a pharmaceutical industry. Alice, keen to contribute to the improvement of public health, agrees to participate in both releases: data targeted by the survey and allowed by Alice to be sanitized, are exported in a privacy-preserving form to the agency. She could have chosen to participate in a single release or even in none, but she knows that her participation will not jeopardize her privacy thanks to the holistic safeguards protecting her health data: the sanitized releases will respect the strongest known privacy models, and the privacy algorithms - involving other participating tokens and a central server - will be executed safely. Neither negligence nor prying eyes may affect the sanitization process anymore since her raw data will never appear in the clear on any central server.

Though illustrated by a medical scenario, our approach is not restricted to the healthcare domain: similar scenarios can be envisioned each time the legislation recognizes the right of the individual to control under which conditions her data is stored and accessed.

1.5 Outline

The recipe of this thesis is based on three ingredients: privacy-preserving data publishing models and algorithms, cryptography, and secure multi-party computation protocols. Chapter 2 presents the background knowledge necessary to understand the approach proposed and positions it with respect to related works. Chapter 3 clearly states the problem tackled in this thesis, by formulating the assumptions made on tokens and the publisher, and the way we propose to revisit the implementation of privacy-preserving data publishing algorithms - including the definition of the security and correctness models used in this work. Chapter 4 details the design of the Protogen suite of protocols, coping with an incrementally stronger adversarial publisher, and validates them experimentally. Chapter 5 describes the Meta-Protocol suite, obtained by generalizing the key findings of Protogen; the chapter is written in a mold similar to the previous chapter. Chapter 6 presents the token-based EHR architecture, and discusses the practical aspects related to a real-life implementation of the Protogen and Meta-Protocol suites. Finally, Chapter 7 synthesizes the contributions made by this thesis, and draws exciting research perspectives.

Chapter 2

Background-Knowledge & Related Approaches

Summary: *This chapter provides the background knowledge necessary for understanding the contributions of this thesis. We start by focusing on the non-interactive* privacy-preserving data publishing *(*PPDP *for short) models and algorithms. We untangle the research path that has led to nowadays' PPDP approaches and underline their intrinsic diversity. Then, we give the background knowledge required for understanding the* cryptographic primitives *used in this work. We emphasize the properties that we expect from these primitives and illustrate them through standard cryptographic algorithms (that current tokens are equipped with). Finally, we overview the approaches related to this thesis. We explain the origin of the cost of the* generic secure multi-party computation *approach (which translates centralized functions representable as a combinatorial circuit to a secure decentralized protocol). Next, we focus on the* specific secure multi-party computation *approaches dedicated to implementing centralized data publishing models and algorithms in decentralized contexts, and show that none of them fulfills simultaneously the* ***Decentralization****,* ***Genericity****, and* ***Generality*** *goals of the thesis. Finally we overview the* token-based secure multi-party computation *and* anonymity-preserving data collection *related approaches.*

Contents

2.1 Introduction

PRIVACY-PRESERVING data publishing models and algorithms aim at providing to *recipients* (e.g., epidemiologists of a governmental agency, statisticians of a pharmaceutical company) a privacy-preserving view of a private *dataset* containing the personal data of a set of *individuals* (e.g., patients of an hospital, customers of a shopping center). The algorithms are *interactive* when they answer continuously to statistical queries posed by the recipients over the dataset; they are *non-interactive* when they produce a sanitized version of the dataset, and deliver it to a recipient. This thesis focuses on the non-interactive approaches to privacy-preserving data publishing. Non-interactive approaches further divide into the *local perturbation* and *centralized publishing* families [Rastogi07]. Traditionally, these two families tackle different settings. The centralized publishing setting usually assumes the existence of a trusted server (e.g., the hospital's server), called the *publisher*, in charge of collecting the individuals' data, executing one or more privacy algorithm(s) on them, and delivering to recipients *sanitized releases*, each protecting the privacy of individuals through a chosen privacy model. On the contrary, the local perturbation setting is free from any trusted server assumption: individuals locally perturb their data (therefore independently) and send this sanitized data directly to the recipients. Local perturbation algorithms originate from the difficulty to get the consent of individuals for collecting their raw data records [Warner65]. In the following, we overload the use of these terms by calling *privacy-preserving data publishing* the non-interactive approaches, and *privacy algorithms and models* the centralized publishing algorithms and models.

The decentralized context of this thesis straightforwardly supports the use of local perturbation algorithms. However, centralized publishing algorithms have been shown to be able to achieve a much better privacy/utility trade-off by using the opportunities offered by the centralization of the complete dataset [Rastogi07]. Hence, this thesis seeks to provide ways of using centralized publishing algorithms in a context where individuals manage their data autonomously. We concentrate on tabular data and leave the investigation of other data types as future work (see Section 7.3 for perspectives).

So far, only the *secure multi-party computation* approach (*SMC* for short) has considered a decentralized approach to centralized publishing. Secure multi-party computation protocols consider a context where a set of parties wishes to compute the output of a function over their data such that no party learns anything significant concerning the others' data. Recently, various works have considered secure multi-party computation protocols on tokens. Less related, the *anonymity-preserving data collection* approach considers that the identifying information is not contained in the data itself but in the communication link (in its largest meaning) - it consequently seeks to collecting data while destroying the association between each respondent and its

response.

This chapter provides the background knowledge necessary for understanding the contributions of this thesis. First, we overview the rich field of privacy-preserving data publishing for sanitizing tabular data. We primarily focus on the centralized publishing family of privacy models and algorithms, and, we briefly look at the local perturbation family, in order to have a clear overall understanding. Second, we present the background necessary for understanding the cryptographic primitives that we use for adapting centralized publishing approaches to the decentralized context of this thesis. Third and last, we position our approach with respect to the secure multi-party computation approaches (with and without tokens) and to the anonymity-preserving data collection approach.

2.2 Non-Interactive Privacy-Preserving Data Publishing

PRIVACY-PRESERVING data publishing models and algorithms are historically strongly tied together - a new model being usually proposed with a proof-of-concept algorithm. Nevertheless, we present them dissociated from each other, favoring thus the understanding of their distinct respective roles. We stress that the goal of this section is to give the background knowledge necessary for understanding the aspects of this thesis related to privacy-preserving data publishing, not to provide a complete survey of the field. Hence, we focus on the key influential models, omitting variations, and tracing a clear path in the (*centralized publishing* family of the) privacy-preserving data publishing models literature for sanitizing tabular data. Then, we explain the major privacy algorithms designed to enforce them. We will describe especially deeply the models and algorithms used along this thesis. We will not present utility metrics since they are orthogonal to the work conducted in this thesis. The conclusive words of the overview outline the great variety of models and algorithms by pointing out the absence of a single universal approach to privacy-preserving data publishing. Finally, we shortly discuss the other family of non-interactive privacy-preserving data publishing models and algorithms: the *local perturbation* approach.

Note that *statistical disclosure control* and *privacy-preserving data mining* are closely related to privacy-preserving data publishing. Statistical disclosure control designates the answer of the *statistics community* to the question of publicly releasing data related to individuals (mainly in an aggregated form), whereas privacy-preserving data publishing originates from the *computer science community*. Privacy-preserving data publishing notably provides deep insights into adversarial models. *Privacy-preserving data mining* can be considered as a super-set of privacy-preserving data publishing [Aggarwal08]; it additionally encompasses techniques for executing data mining algo-

rithms on sanitized data, secure multi-party computation protocols tailored for computing data mining outputs (e.g., classifiers), or techniques for verifying that the data mining outputs do not violate privacy. In the following, we focus on privacy-preserving data publishing, referring neither to statistical disclosure control nor to privacy-preserving data mining anymore, and invite the interested reader to consult the surveys [Adam89] for statistical disclosure control and [Verykios04, Aggarwal08] for privacy-preserving data mining.

2.2.1 Prominent Centralized Publishing Privacy Models

The k-Anonymity Model

The disclosure cases presented in the introduction highlight the possibility of linking records to individuals based on data other than *direct identifiers*. In the case brought to light by Sweeney, the medical record of the governor of Massachusetts is linked back to the governor based on a set of attributes that is identifying for some individuals but not all, i.e., the social information associated to the medical record (in the medical dataset) was associated to the governor's identity in an external data source (the voter list). A set of attributes that may be identifying is called a *quasi-identifier* of the dataset [Sweeney02] (Definition 1). A dataset may contain several quasi-identifiers; we denote $\mathcal{QI}$ the set of quasi-identifiers hereafter.

Definition 1 (Quasi-Identifier *(from [Machanavajjhala06])*)**.** Let $\mathcal{D}$ be the initial dataset, i.e., a table whose schema consists of the attributes $\mathcal{A} = \{A_1, \ldots, A_n\}$, and let Ω represent the population from which $\mathcal{D}$ is sampled (i.e., $\mathcal{D} \subset \Omega$). The set of attributes $\{Q_1, \ldots, Q_j\} \subseteq \mathcal{A}$ is called a quasi-identifier if these attributes can be linked with external data to uniquely identify at least one individual in the general population Ω.

In the medical disclosure case presented above, $\mathcal{D}$ is the medical dataset, Ω the Massachusetts's population, and $\{gender, birthdate, zipcode\}$ is the quasi-identifier used by the attack. The voter list is the background knowledge of the attacker.

The above attacks, called *linking attacks* [Chen09a], could be thwarted by a mere pseudonymization scheme if the quasi-identifiers were removed from the dataset, similarly to direct identifiers. But what would be the resulting information loss? To preserve utility while not thwarting privacy, Samarati and Sweeney proposed through the k-Anonymity privacy model [Samarati98, Sweeney02] (Definition 2) to blur the link between individuals and their corresponding records rather than to delete it completely. The k-Anonymity model states that, in the sanitized dataset, each record

must share its quasi-identifier with (at least) $k-1$ other records. Consequently, whatever the external data available to the attacker concerning the quasi-identifier, he will not be able to link a given individual to a given k-anonymous record with a probability greater than $1/k$.

Definition 2 (k-ANONYMITY *(from [Samarati98, Sweeney02, Machanavajjhala06])*)**.** Let $\mathcal{V}$ be the sanitized dataset and $\mathcal{QI}$ denote the set of all quasi-identifiers associated with it. $\mathcal{V}$ satisfies k-ANONYMITY if for every record $r \in \mathcal{D}$ there exist $k-1$ other records $r_1, r_2, \ldots, r_{k-1} \in \mathcal{D}$ such that $r[QI] = r_1[QI] = \ldots = r_{k-1}[QI]$ for all $QI \in \mathcal{QI}$, where $r[QI]$ denotes the projection of r on the attributes of QI.

Few records *naturally* have a quasi-identifier value shared by $k-1$ other records. The k-ANONYMITY model is usually enforced by *coarsening* in a controlled manner the quasi-identifier values such that each record becomes indistinguishable from (at least) $k-1$ other records with respect to their (degraded) quasi-identifier (see Section 2.2.2 for more details). We call the set of sanitized records sharing the same coarsened quasi-identifier value an *equivalence class* (we define this notion formally below when presenting the generalization-based algorithms).

The ℓ-Diversity Principle

The k-ANONYMITY privacy model prevents linking a given individual to less than k records in a k-anonymous dataset. However, attributes that do not appear in the quasi-identifier are not taken into account, even though they are highly sensitive (e.g., medical diagnosis, salary, political orientations). In [Machanavajjhala06, Machanavajjhala07], the authors aim at protecting the association between individuals and these *sensitive values*. To simplify the discussion, we follow the literature and assume in the following that the attributes of the dataset are made of a *single quasi-identifier* (i.e., consisting of the union of the dataset's quasi-identifiers) and a *single sensitive attribute*.

They formulate the BAYES-OPTIMAL PRIVACY model by defining precisely the impact of a sanitized release on the adversarial belief. They model the (worst-case) adversarial *prior belief* as the exact joint distribution over the quasi-identifier values and sensitive values of the whole population Ω. Based on this distribution, the adversary's prior belief about the likeliness of an association between a given quasi-identifier q and a given sensitive value s is the conditional probability to observe s associated to q. Second, they compute the adversarial belief *impacted* by the sanitized release, called the *posterior belief*, based on Bayesian probabilities. Finally, they define a *disclosure* as being a significant difference between the posterior and prior adversarial beliefs. Defining disclosures by comparing the adversary's prior knowledge to its posterior knowledge

is called the *uninformative principle* [Machanavajjhala06]. The uninformative principle is rooted in Dalenius's early definition of statistical disclosure [Dalenius77] and has inspired most influential modern privacy models [Evfimievski03, Chen07, Rastogi07, Machanavajjhala07, Li10b]. As we will see in the following, the uninformative principle is not the only privacy paradigm currently existing [Dwork06a].

Definition 3 (Uninformative Principle *(from [Machanavajjhala06])*)**.** The sanitized release should provide the adversary with little additional information beyond his prior knowledge. In other words, there should not be a large difference between the prior and posterior adversarial beliefs.

Though appealing for its soundness, the Bayes-Optimal Privacy model is impractical notably because (1) the publisher does not necessarily know the exact complete joint distribution between quasi-identifier and sensitive values, (2) the publisher does not necessarily know what the adversary knows, and (3) furthermore there may be multiple adversaries with various background knowledge.

The authors identify two conditions under which the Bayes-Optimal Privacy model is thwarted: (1) the sensitive values associated to the shared quasi-identifier value of a given record are not *diverse* enough (e.g., in a set of medical records sharing the same quasi-identifier, if (almost) all are associated to the "Cancer" disease, the attacker learns that these quasi-identifiers have a high probability to be associated to this sensitive value), and (2) if the adversary knows that a given individual is *not* associated to some sensitive values, he is able to filter them out from the sensitive values associated to the individual's shared quasi-identifier value. Consequently, the authors propose the ℓ-Diversity principle (Definition 4) as a practical alternative to the Bayes-Optimal Privacy model, stating that each shared quasi-identifier value must be associated to a diverse enough set of sensitive values.

Definition 4 (ℓ-Diversity (from [Machanavajjhala06]))**.** An equivalence class is ℓ-diverse if it contains at least ℓ *"well-represented"* values for the sensitive attribute. A table is ℓ-diverse if every equivalence class is ℓ-diverse.

The ℓ-Diversity principle can be instantiated in as many ways as the notion of *"well-represented"* can be. Well-known instantiations encompass the Entropy ℓ-Diversity (stating informally that, in each class, the entropy of each sensitive value must exceed a lower bound), Recursive (c, ℓ)-Diversity (stating informally that, in each class, the most frequent sensitive values must not appear too frequently, and the least frequent sensitive values not too rarely - this guarantees a smooth decrease of the privacy protection with respect to an adversary able to filter out an increasing number of sensitive values (e.g., Alice knows that Bob does not have "Flue")), and a "folkloric"

instantiation of ℓ-DIVERSITY often used in the literature (merely requiring each class to contain at least ℓ distinct sensitive values).

The Closeness Principle

The authors of the CLOSENESS privacy principle (instantiated by the t-CLOSENESS [Li07a] and (n,t)-CLOSENESS [Li10b] models) criticize the relevancy of the ℓ-DIVERSITY principle when sanitizing datasets in which the distribution of the sensitive attribute is highly skewed. They criticize the utility guarantees by observing that ℓ-DIVERSITY is hard to satisfy when the sensitive attribute is already not diverse in the complete dataset, leading to large information loss in the sanitized release. They criticize the privacy guarantees by questioning the disclosure incurred by an equivalence class which is ℓ-Diverse but whose internal sensitive value distribution is far from the whole dataset's distribution. For example, what about the privacy of individuals whose quasi-identifier values appear in a 3-Diverse class containing 30% of "Cancer" diseases, whereas only 1% of "Cancer" appear in the complete dataset?

The major change proposed by the t-CLOSENESS principle [Li07a] is to consider as *an auxiliary source of adversarial knowledge* the distribution of the sensitive attribute in the complete dataset. Moreover, this is the only source of adversarial knowledge known with certainty by the publisher. These considerations translate to the uninformative principle (Definition 3) as follows: (1) the adversarial prior knowledge is the distribution of the sensitive attribute in the complete dataset, (2) the adversarial posterior knowledge is the distribution of the sensitive attribute in each equivalence class of the sanitized release, and consequently (3) a disclosure occurs when the distance between the distribution of a class and the distribution of the complete dataset exceeds the threshold t. Forcing the distribution in each class to be close to the global distribution results however in severe information losses. To cope with this drawback, the (n,t)-CLOSENESS principle [Li10b] relaxes the distance computation by not requiring it anymore to relate to the distribution of the sensitive attribute in the complete dataset but to the distribution in subsets of the dataset that contain at least n records (see [Li10b] for details).

Specifying the Adversarial Knowledge

The ℓ-DIVERSITY and CLOSENESS principles assume a certain form of adversarial knowledge (in addition to the quasi-identifier values of target individuals) against which they provide a quantifiable amount of protection: the CLOSENESS principle restricts the adversarial knowledge to be the sensitive attribute distribution in the sanitized release, and the ℓ-DIVERSITY principle considers an adversary able to filter out some sensitive

values from equivalence classes (in other words, the adversarial knowledge is made of *negation statements* over sensitive values). The works of Martin & al [Martin07] and Chen & al [Chen07, Chen09b] further specify - through languages based on logic sentences - *the amount of adversarial knowledge* (and not the precise content of this knowledge) against which the sanitized release has to be protected.

The language proposed in [Martin07] assumes adversaries able to formulate a conjunction of at most k logic sentences over the associations between individuals and sensitive values whatever the specific content of the conjunctions. This language, denoted $\mathcal{L}^k_{basic}$, is shown to permit the formulation of general predicates on the table, and consequently to model arbitrarily powerful adversaries (bounded by the user-defined parameter k). The equivalence classes in the sanitized release are such that, given the expected number of adversarial logic sentences, the probability of associating *any individual to any sensitive value* remains lower than a predefined threshold. This is called the (c, k)-SAFETY privacy model. In [Chen07, Chen09b], Chen & al propose a complementary language modeling the amount of adversarial knowledge - about the association between *any* individual i and a specific sensitive value σ - along three dimensions: (1) knowledge about i (e.g., Bob does not have σ), (2) knowledge about other individuals (e.g., Charlie has σ), and (3) knowledge about the relationship between i and other individuals (e.g., if Maria has σ then Bob too). The expected amount of adversarial knowledge must thus be specified by the publisher, along these axes, for each sensitive value (authors only consider a subset of the sensitive values to be really sensitive - e.g., a "Cold" diagnosis may not be sensitive whereas an "HIV" may be). The equivalence classes in the sanitized release are such that, for all the (really) sensitive values σ, given the expected amount of adversarial knowledge about σ, the probability of associating *any individual to* σ is lower than a predefined threshold. This is called the 3D-PRIVACY model. The (c, k)-SAFETY and 3D-PRIVACY languages are complementary *in practice* because each of them permits *practically* the expression of distinct forms of adversarial knowledge (though the (c, k)-SAFETY language is in theory more expressive). We refer the interested reader to [Chen09b, Chen09a] for a deeper comparison.

The (d, γ)-Privacy Model

With the (d, γ)-PRIVACY model [Rastogi07] we leave the (now familiar) ground of models based on equivalence classes; the distinction between quasi-identifier and sensitive attributes is not required anymore: all the attributes of the dataset are treated identically. Faithful to the uninformative principle (Def. 3), the (d, γ)-PRIVACY model limits the distance between the adversary's expected prior knowledge and its posterior knowledge. Another significant difference between this model and the above ones is that *it*

characterizes the sanitization algorithms. Indeed, to be (d, γ)-private, a sanitization algorithm must not be deterministic; it must involve a controlled amount of *randomness* such that its output distribution disallows significant adversarial knowledge gains.

Modeling the Problem. The initial dataset $\mathcal{D}$ consists in a set of $n_{\mathcal{D}}$ unique and independent records taking their values over the finite definition domain *dom* (i.e., the cross-product of their attributes's domains). Whatever the way the adversary obtained his prior knowledge, the result is a probability distribution over the possible initial datasets, denoted $\mathtt{Pr}_1$, and such that $\sum_{\mathcal{D} \subseteq dom} \mathtt{Pr}_1(\mathcal{D}) = 1$. Such prior knowledge defines the probability $\mathtt{Pr}_1(r)$ of each possible record r from the domain *dom* to be part of the actual initial dataset $\mathcal{D}$: $\mathtt{Pr}_1(r) = \sum_{\mathcal{D} \subseteq dom : r \in \mathcal{D}} \mathtt{Pr}_1(\mathcal{D})$. The initial dataset $\mathcal{D}$ is sanitized by a *randomized* sanitization algorithm in order to produce the sanitized release $\mathcal{V}$ defined over the same definition domain *dom*. We will present sanitization algorithms below; for the moment we only need to know that the (d, γ)-Privacy models the algorithm's output as a probability distribution over the possible sanitized releases. The output distribution given the input dataset $\mathcal{D}$ is denoted $\mathtt{Pr}_2^{\mathcal{D}}$, and is such that $\sum_{\mathcal{V} \subseteq dom} \mathtt{Pr}_2^{\mathcal{D}}(\mathcal{V}) = 1$. Contrary to $\mathtt{Pr}_1$, the publisher controls $\mathtt{Pr}_2$. Finally, when accessing the sanitized release, the adversary forms his posterior knowledge, denoted $\mathtt{Pr}_{12}(r|\mathcal{V})$, based on his prior $\mathtt{Pr}_1$ and the (public) distribution of the sanitization algorithm $\mathtt{Pr}_2$: $\mathtt{Pr}_{12}(r|\mathcal{V}) = \sum_{\mathcal{D} \subseteq dom : r \in \mathcal{D}} \mathtt{Pr}_1(\mathcal{D}) \cdot \mathtt{Pr}_2^{\mathcal{D}}(\mathcal{V})$.

The (d, γ)-Privacy Model. Similarly to above approaches, the (d, γ)-Privacy model tackles adversaries that have an expected amount of prior knowledge specified by the publisher (this is the parameter d); this expected amount of knowledge is actually a bound on the prior knowledge. The model does not protect against adversaries more powerful than expected; it considers that records for which the prior exceeds d are already known to the adversary. Hence: $\forall r \in \mathcal{D}$, $\mathtt{Pr}_1(r) \leq d$ or $\mathtt{Pr}_1(r) = 1$. We are now ready to define the (d, γ)-Privacy model (Definition 5).

Definition 5 ((d, γ)-Privacy *(from [Rastogi07])*)**.** An algorithm is (d, γ)-private if the following holds for all adversarial prior $\mathtt{Pr}_1$, sanitized release $\mathcal{V}$, and record r s.t. $\mathtt{Pr}_1(r) \leq d$:

$$\frac{d}{\gamma} \cdot \mathtt{Pr}_1(r) \leq \mathtt{Pr}_{12}(r|\mathcal{V}) \leq \gamma$$

Loosely speaking, the (d, γ)-Privacy model limits the adversarial information gain: the posterior knowledge must not increase "too much" (i.e., as stated by the right member, it must remain lower than the user-defined threshold γ), and must not decrease "too much" (i.e., as stated by the left member, it must remain higher than a fraction of the prior knowledge). Any (d, γ)-private algorithm satisfies the Differential Privacy model introduced next.

Differential Privacy

While originally focused on the interactive answering of aggregate queries posed over a static private dataset [Dwork06b, Dwork06a], the Differential Privacy approach has received increasing attention from the database and security research community [Korolova09, Dwork09, Rastogi09, Inan10, Hay10, Li10a, Hardt10, Dwork10, Rastogi10, Ding11, Kifer11] as well as from a more general computer science audience [Greengard08, Narayanan10, Dwork11]. Although this thesis does not consider interactive privacy-preserving data publishing approaches, we mention Differential Privacy for two main reasons. First, Differential Privacy defined a groundbreaking approach to privacy. Unlike the models presented above, it is not based on limiting the difference between the adversarial (expected) prior and posterior knowledge (i.e., the uninformative principle) but on limiting *the impact of the participation of any single record on the output of the sanitization algorithm*. In itself, this new privacy paradigm would warrant the study of Differential Privacy. Second, although Differential Privacy was initially designed for an interactive context, it has been shown to be related to non-interactive privacy-preserving data publishing models (e.g., (d, γ)-private algorithms satisfy Differential Privacy [Rastogi06, Rastogi07]) and to the uninformative principle [Rastogi09]. Hence, the Differential Privacy approach has an impact much broader than its original intent.

Similarly to (d, γ)-Privacy, Differential Privacy characterizes sanitization algorithms. It requires them to be randomized, and such that, given two input datasets differing by a single tuple, their corresponding output distributions do not differ substantially. Two flavors of Differential Privacy are prominent [Dwork06b, Dwork06a]; we give in Definition 6 the one to which (d, γ)-Privacy directly relates, namely, ϵ-Indistinguishability [Dwork06b] (also called Bounded Differential Privacy in [Kifer11]).

Definition 6 (ϵ-Indistinguishability *(from [Dwork06b, Kifer11])*)**.** A randomized algorithm $\mathbb{A}$ satisfies ϵ-Indistinguishability if $\texttt{Pr}(\mathbb{A}(\mathcal{D}_1) \in S) \leq e^{\epsilon} \cdot \texttt{Pr}(\mathbb{A}(\mathcal{D}_2) \in S)$ for any set S and any pair of databases $\mathcal{D}_1, \mathcal{D}_2$ where $\mathcal{D}_1$ can be obtained from $\mathcal{D}_2$ by changing the value of exactly one tuple.

The second flavour is called ϵ-Differential Privacy [Dwork06a]; it differs from ϵ-Indistinguishability in that it considers that $\mathcal{D}_1$ is obtained from $\mathcal{D}_2$ by inserting or removing exactly one tuple (rather than changing the value of one tuple).

The intuition behind Differential Privacy is that a sanitized output does not hurt the privacy of an individual if what can be learned with the individual's participation can also be learned without her participation. For example, an individual attempting to subscribe a car insurance to a company consulting differentially private

statistics, does not have more risk of denial if she participates in the statistics than if she does not. Contrary to models based on the uninformative principle, DIFFERENTIAL PRIVACY frees the publisher from explicitly formulating the expected power of the adversary (though, as explained below in Sec. 2.2.3, adversarial assumptions are actually implicit).

Towards a Unified Theory of Privacy and Utility

A large amount of privacy models has been proposed during the last decade based on various assumptions about adversarial abilities and various definitions of disclosure (see the surveys [Chen09a, Fung10]). Recent efforts have started to relate them together by drawing encompassive adversarial models [Machanavajjhala09, Rastogi09] on the one hand and rigorously designing the foundations of a unified theory of privacy and utility [Kifer10] on the other hand. Note that other works have considered a unifiying approach, such as [Bertino05, Bertino08] that draw objective comparison criteria for privacy-preserving data mining models and algorithms.

Machanavajjhala & al define in [Machanavajjhala09] a powerful and generic model of the adversarial knowledge and reasoning. They observe that modeling the adversarial reasoning as a probability distribution over the possible records from which the actual records are drawn independently is incorrect because it misses the uncertainty of the adversary concerning his own prior as well as the dependencies between records. They propose an alternative model consisting of a distribution over an infinite set of probability vectors, each vector being a (more or less) probable distribution from which the actual records can have been independently drawn. This model, shown to overcome the above deficiencies, uncovers a new adversarial space whose boundaries depend notably on the willingness of the adversary to learn from the sanitized release (e.g., an adversary that has a strong prior knowledge based on sound statistical studies is unlikely to change his belief easily). The authors show how the adversarial assumptions of previous models are covered (e.g., (c, ℓ)-DIVERSITY, DIFFERENTIAL PRIVACY, t-CLOSENESS), and identify new classes of adversaries. To tackle these latter, the authors propose the ϵ-PRIVACY model inspired from DIFFERENTIAL PRIVACY and also show how ϵ-PRIVACY relates to previous privacy models.

In [Rastogi09], Rastogi & al tackle the problem of answering interactive queries posed over private social network graphs. An interesting result from this work is the bridge that it builds between the models following the uninformative principle, also called the *adversarial privacy* models in [Rastogi09], and the DIFFERENTIAL PRIVACY model: the authors show that ϵ-INDISTINGUISHABILITY actually corresponds to a specific instantiation of the *adversarial privacy* model that they propose. We refer the reader to [Rastogi09] for details.

A theory of privacy and utility is currently emerging. In [Kifer10], Kifer & Lin initiate the study of guidelines for paving the way for a principled design of privacy models and algorithms. These guidelines consist of two *privacy axioms*, that state the privacy requirements to be expected from any privacy model and algorithm, and one *utility axiom*, that states the utility requirements to be expected from any privacy algorithm. In order to make the approach concrete, [Kifer10] generalizes and relaxes the Differential Privacy model based on the axioms proposed. However, far from consisting in a one-size-fits-all approach, the axioms are not tied to any specific privacy or utility model. The benefits of this approach range from the most practical aspects (e.g., enlightened choice of the privacy model best suited to a given setting) to the most theoretical ones (e.g., understanding the relationships between privacy paradigms).

As a conclusive word about unifying approaches, we note that they do not aim at designing a single one-size-fits-all privacy model but seek to provide tools for understanding the existing privacy models and for designing the future ones.

2.2.2 Prominent Centralized Publishing Privacy Algorithms

Generalization-Based Algorithms

The Basic Generalization Action. We start the explanation of generalization-based algorithms by presenting the basic action which grounds them: *generalization*. Generalization is simply the action of coarsening a value: the precise value - categorical (e.g., a country) or numerical (e.g., an age) - is replaced by a range (e.g., a group of countries, a range of ages). Generalization is also called *recoding* in the statistics vernacular. Some semantic information about the value to generalize is needed in order to know to which general value(s) it can actually be coarsened (e.g., expressing the fact that "Paris" can be generalized to "France"). Moreover, this semantic information must be organized hierarchically, some generalized values being *more general* than others (e.g., "Paris" is less general than "France", which is less general than "Europe"). *Value generalization hierarchies* (VGH for short) express this semantic knowledge. As depicted in Figure 2.1, a VGH is a tree whose leaves are the domain values (e.g., if the domain contains European cities, then "Paris" is a leaf), whose root is the most general value (e.g., often set to "Any"), and whose nodes on the path from the root to a leaf represent decreasing levels of generalization. Nodes in a generalization hierarchy are called *generalization nodes*. Following the literature's convention, we denote $\succeq$ the generalization relationship between two nodes where the left node generalizes the right one. The reverse relationship is called *specialization* and denoted $\preceq$. Value Generalization Hierarchies for numerical definition domains are similar. Note that these hierarchies can be static (i.e., predefined by an expert) or dynamic (i.e., computed

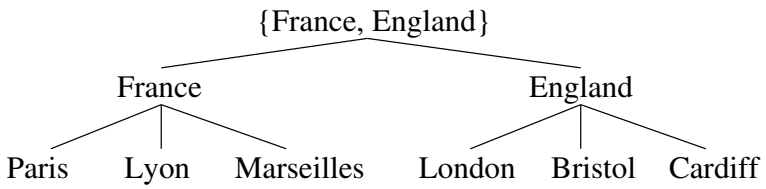


Figure 2.1: Example of a Value Generalization Hierarchy

on the fly by the generalization algorithm - see the MONDRIAN algorithm below).

Equivalence Classes. The attributes in the quasi-identifier are usually those considered for generalization, but nothing precludes generalizing sensitive values (e.g., as proposed in [Tian11]). The objective of generalizing the quasi-identifier values is to form non-overlapping (multi)sets of records, such that all records in the same set share the same generalized quasi-identifier. For simplicity, the generalized quasi-identifier (made of one generalized value per attribute) is called the generalization node. A set of records with its generalization node is called an *equivalence class*. Depending on the privacy model to enforce, equivalence classes will have to fulfill different constraints on the number of records they contain (e.g., k-ANONYMITY requires each class to contain at least k records), or on the distribution of sensitive values (e.g., folkloric ℓ-DIVERSITY requires each class to contain at least ℓ distinct sensitive values). Thus, generalization-based algorithm input the initial dataset and output a set of non-overlapping equivalence classes, denoted $\mathcal{E}$ in the following.

Definition 7 (Equivalence Class)**.** An *equivalence class* $\mathcal{E}_i \in \mathcal{E}$ consists of a *content*, i.e., a multi-set of records, denoted $\mathcal{E}_i.R$, and of a *generalization node*, denoted $\mathcal{E}_i.\gamma$, such that for all record $r \in \mathcal{E}_i.R$, for all attributes $Q_j \in QI$, then $r[Q_j] \preceq \mathcal{E}_i.\gamma[Q_j]$.

In the following, we overload the notations by writing $r[QI] \preceq \mathcal{E}_i.\gamma$ if each generalized value in the generalization node $\mathcal{E}_i.\gamma$ generalizes the corresponding value in r's quasi-identifier.

Finding the Best Generalized Release is Hard. Any dataset could be trivially generalized to a single equivalence class whose generalization node consists of the roots of all the value generalization hierarchies (e.g., replace each quasi-identifier by {"Any", ..., "Any"}). This trivial algorithm is similar to simply suppressing the quasi-identifier in all records. But it is possible to do "better"; generalization-based algorithms seek the generalized release that satisfies the given privacy model and preserves as much utility as possible (e.g., choosing generalization nodes as close as possible to the leaves in the value generalization hierarchies). In practice, given an initial dataset, there is not one single possible generalized release but many (various levels of generalization, several attributes in the quasi-identifier). *How is the* optimal *generalized release found?*

Finding the optimal generalization is hard due to the combinatorial explosion in the number of possible generalized releases [Meyerson04, Bayardo05]. The question thus becomes: *How is a* good *generalized release found?* There are many possible answers to this question, as shown by the number and diversity of the generalization-based algorithms proposed, e.g., [Samarati01, Sweeney02, Meyerson04, Wang04, Bayardo05, Fung05, Xu06, LeFevre06, Aggarwal06, Ghinita07, Nergiz07a, Park07, Ghinita09].

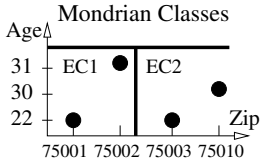
Global/Local Recoding. A generalization-based algorithm falls either into the *global recoding* or the *local recoding* families. With global recoding algorithms (e.g., [LeFevre05, LeFevre06]), the generalization nodes of the set of equivalence classes *partition the dataset*: a given record specializes a *single* generalization node. For example, all the records that specialize the generalization node {Location = "France", Age = "[20, 30]"} will appear into the node's equivalence class. This is not necessarily the case with local recoding algorithms (e.g., [Xu06]) where generalization nodes may *overlap* - i.e., the intersection between the two sets of leaves that they respectively generalize is not empty. In other words, a given record may specialize several generalization nodes. In this case, the choice of the equivalence class in which the record will appear depends on the algorithm. For example, a set of equivalence classes may contain the generalization nodes γ_1 ={Location = "France", Age = "[17, 25]"} and γ_2 ={Location = "{France, England}", Age = "23"}. The record r ={Location = "Paris", Age = "23"} specializes both γ_1 and γ_2 (i.e., $\gamma_1 \cap \gamma_2 \neq \varnothing$). The choice of the equivalence class in which r will appear depends on the underlying generalization-based algorithm.

The global recoding family can further be divided into *single-dimensional recoding* (e.g., [LeFevre05]) and *multi-dimensional recoding* (e.g., [LeFevre06]). With single-dimensional recoding, the domain of each attribute (i.e., each single dimension of the quasi-identifier) is uniformly generalized. For example, in the Location attribute, all values, e.g., "Paris" will be generalized to, e.g., "France"; in the Age attribute, all values, e.g., "23" will be generalized to, e.g., "[20, 30]". Multi-dimensional recoding permits additional flexibility by generalizing the cross-product of domains (i.e., the multi-dimensional domain). For example, the record, e.g., {Location = "Paris", Age = "23"} will be generalized to, e.g., {Location = "France", Age = "[20, 30]"}, while the record, e.g., {Location = "Paris", Age = "35"} will be generalized to, e.g., {Location = "{France, England}", Age = "[34, 35]"}. Indeed, these two generalization nodes *do not overlap* (any record generalized by the one cannot be generalized by the other) even though the single-dimensional node "{France, England}" overlaps with the other node "France" (both generalize the leaves below "France").

Let us now briefly and intuitively describe the MONDRIAN multi-dimensional heuristic [LeFevre06] (Chapter 4 often refers to it for illustration purposes). MONDRIAN considers the definition domain of the quasi-identifier as a multi-dimensional space, and

Raw Data

Zip	Age	Diag.
75001	22	Cold
75002	31	Flue
75003	22	Cold
75010	30	HIV



2–Anonymous Data

Zip	Age	Diag.
[75001, 75002]	[22, 31]	Cold
[75001, 75002]	[22, 31]	Flue
[75003, 75010]	[22, 31]	HIV
[75003, 75010]	[22, 31]	Cold

Figure 2.2: Example of a MONDRIAN Space Partitioning Forming 2-anonymous Equivalence Classes

each quasi-identifier value in the dataset as a point in this space. Figure 2.2 depicts a raw dataset, the corresponding multi-dimensional space considered by MONDRIAN, and the 2-anonymous equivalence classes resulting from the MONDRIAN algorithm (sketched below). The raw dataset is made of three attributes: a quasi-identifier consisting of the Age and Zipcode attributes (the two axes of the multi-dimensional space), and a sensitive attribute consisting of the Diagnosis attribute. The MONDRIAN algorithm processes by recursively partitioning the multi-dimensional space. The first iteration starts with a single partition (covering all the quasi-identifier values in the dataset), elects a dimension (e.g., the one in which the distance between the min and max values is the greatest), computes its median, and splits the partition along the median, thus forming two partitions. The next iteration considers the resulting partitions, and follows the same process to split them. The partitioning process stops when no partition can be further split, i.e., a split would result in a partition not satisfying the privacy model (e.g., for k-ANONYMITY a partition which would contain less than k records). In Fig. 2.2, the sanitized release must satisfy k-ANONYMITY with $k = 2$. MONDRIAN splits the partition that covers the entire space, along the Zipcode dimension. The resulting partitions cannot be further split because this would result in partitions containing less than 2 records so would not permit the generation of 2-anonymous equivalence classes. Note that MONDRIAN can be extended to include privacy constraints on the distribution of sensitive values within each class in order to enforce privacy models such as, e.g., ℓ-DIVERSITY or ϵ-PRIVACY.

Privacy Models Enforced. Generalization-based algorithms (notably the INCOGNITO [LeFevre05] and MONDRIAN [LeFevre06] algorithms) have been used to enforce various privacy models including but not limited to k-ANONYMITY (their initial target privacy model), ℓ-DIVERSITY, t-CLOSENESS, 3D-PRIVACY, ϵ-PRIVACY. These latter are enforced by specifying constraints that include the distribution of sensitive values within each equivalence class.

Bucketization-Based Algorithms

The Bucketization algorithm was initially proposed by Xiao & Tao in [Xiao06] to produce ℓ-diverse equivalence classes (1) with less information loss and (2) more efficiently than generalization-based algorithms. Bucketization assumes that a single attribute is sensitive. To tackle Point (1), Xiao & Tao propose to blur the association between quasi-identifier and sensitive values by forming subsets of records (as described below), assigning an identifier to each subset, and simply releasing quasi-identifier and sensitive values in two distinct tables. The association between subsets of quasi-identifier and sensitive values is maintained, in a degraded form, by the subsets' identifiers. This way of publishing data is called Anatomy; we note its similarity with the Permutation approach [Zhang07b]. For simplicity, we also call these subsets equivalence classes, where the generalization node is the subset's identifier. The Bucketization algorithm is in charge of forming the subsets of records efficiently (Point (2)). It starts by distributing the initial records into a set of *buckets* according to their sensitive values (one bucket per value), and keeps a counter of the number of records per bucket. It is then able to form the subsets of records by recursively picking ℓ records in the ℓ buckets that have the highest number of sensitive values. Finally, the records that remain (at most $\ell - 1$) are allocated to subsets that do not already contain their sensitive values. Note that the Bucketization algorithm makes no use of the quasi-identifier values at all when forming the equivalence classes.

The SABRE algorithm [Cao11] revisits the Bucketization algorithm to enforce the t-Closeness privacy model. Loosely speaking, the t-Closeness privacy model requires the distribution of sensitive values within each equivalence class to be *close* to the global distribution in the complete dataset (see above for more details). This global distribution is precisely reflected in the buckets formed by the Bucketization algorithm. However, the t-Closeness's requirement incurs a severe information loss. In order to facilitate the satisfaction of this requirement, SABRE permits the generalization of the sensitive values. To cope with this additional flexibility the formation of buckets follows the generalization hierarchy of the sensitive value. Each resulting bucket contains the records specializing its generalized sensitive value. The buckets reflect together a distribution respecting the t-Closeness requirement. Equivalence classes are formed by redistributing the records now stored into buckets proportionally to the buckets's sizes (buckets are already t-close to the global distribution). The resulting classes consequently satisfy t-Closeness.

We note that the use of a bucketization-based algorithm was not only studied for ℓ-Diversity and t-Closeness, but also for (c, k)-Safety [Martin07].

$\alpha\beta$-Algorithm

The $\alpha\beta$-Algorithm [Rastogi06, Rastogi07] was initially designed to enforce the (d, γ)-Privacy model. It does not rely on the assumption of the existence of quasi-identifier and sensitive values, but treats equally all the attributes of the dataset. It considers a dataset made of $n_{\mathcal{D}}$ unique records; the schema consists in the attributes $\{A_1, \ldots, A_m\}$ defined over the finite domains $\{dom_1, \ldots, dom_m\}$. Each record in $\mathcal{D}$ is thus defined over the domain of all records, denoted dom, resulting from the cross-product of the attributes's domains. The number of records in dom is denoted n_{dom}. The utility of the $\alpha\beta$-Algorithm is formally defined with respect to *counting queries*.

Let us first assume that the parameters α and β have been set. The $\alpha\beta$-Algorithm forms the sanitized release $\mathcal{V}$ by randomly sampling the initial dataset $\mathcal{D}$ and by injecting in $\mathcal{V}$ records generated randomly. It is thus made of two steps: (1) the α-step inserts in $\mathcal{V}$ each record from $\mathcal{D}$ with probability $\alpha + \beta$ and (2) the β-step inserts in $\mathcal{V}$ each record from $(dom - \mathcal{D})$ with probability β. Rather than scanning the full domain, the β-step consists in, first, computing the number of random records to generate - denoted $n_{\mathcal{R}}$ - by sampling the Binomial distribution parametrized by $n_{dom} - n_{\mathcal{D}}$ trials and a success probability β, and, second, in generating $n_{\mathcal{R}}$ unique records not in $\mathcal{D}$ by sampling $(dom - \mathcal{D})$ uniformly. The $\alpha\beta$-Algorithm is shown to enforce the (d, γ)-Privacy model provided that the α and β parameters have been chosen according to the d and γ privacy requirements. We refer the interested reader to [Rastogi07] for more details.

2.2.3 A *One-Size-Fits-All* Privacy Approach is Chimerical

The variety of privacy models and algorithms results from a harsh *break-and-propose* cycle from the research community. Literature indeed suggests that no privacy model/algorithm resists to a long term scrutiny with respect to its founding assumptions (concerning the underlying dataset, the adversarial model, and the recipients's data usage).

The Break-and-Propose Cycle. The k-Anonymity, ℓ-Diversity, and t-Closeness privacy models enforced by generalization-based algorithms have been questioned on their ability to achieve a privacy/utility trade-off better than a trivial sanitization scheme in the context of a classification task over a specific input dataset [Brickell08]. However, Li & Li claim in [Li09] that these results come from a misconception of the privacy and utility notions and propose a methodology expected to model their relationship better based on a financial model.

The attempt of deterministic algorithms to produce (close to) optimal sanitized releases has been shown to open the door to algorithm-based disclosures [Wong07,

Zhang07a, Jin10, Cormode10, Xiao10], leading to privacy models and algorithms resistant to these breaches.

Kifer proposes in [Kifer09] to *revisit* (and not to discard) the sanitization approaches that are based on the (common) assumptions that (1) given a sanitized release any initial dataset is equally likely (called the *random-world* assumption), (2) records are generated independently and identically from a distribution known by the adversary (called the i.i.d assumption), (3) and records are independent. Indeed, these assumptions underestimate the adversarial abilities in reconstructing the initial dataset. This statement is demonstrated through the analysis of a sanitized release that follows Anatomy and satisfies ℓ-Diversity.

In [Kifer11], Kifer & Machanavajjhala clarify the underlying assumptions of the Differential Privacy model. Most notably (for the current section), they (1) propose a variation of Differential Privacy that gets rid of assumptions concerning data but provides limited utility, and (2) show that Differential Privacy does not protect against any type and amount of adversarial background knowledge. This work suggests that privacy models are not absolute objects disconnected from any practical setting: they are always grounded on adversarial and data assumptions.

Finally, the recent work by Cormode [Cormode11] describes a simple attack on the Differential Privacy model, showing that even the current "gold standard" is not an absolute solution to privacy-preserving data publishing.

Conclusive Statement. Does this mean that privacy-preserving data publishing is *unachievable*? We rather believe that the wide variety of usage from data recipients, and trust put into them, is actually a call for a wide variety of models and algorithms that are based on different assumptions about the underlying data, the expected data usage, and the expected adversarial model. The unified theory of privacy and utility - that is currently emerging - actually witnesses this need of diversity (see Section 2.2.1.0) and advances the understanding of the relationships between the various privacy models and algorithms (practical tools such as TIAMAT [Dai09] additionally permit a direct comparison of releases sanitized by different privacy models/algorithms/parameters in order to favor enlightened choices).

2.2.4 A Short Word on Local Perturbation Approaches

The *local perturbation* family consists of the privacy-preserving data publishing approaches that let each individual perturb his own data independently from any other's before sending it to the publisher (or directly to data recipients). Although the flexibility in the sanitization operation is reduced compared to centralized publishing approaches, resulting in decreased utility of the sanitized datasets, local perturbation

approaches remove the need to trust a central publisher: this is a way to increase the trust of individuals in the privacy-preserving data publishing system, thus to lower their resistance to participating truthfully in it [Warner65], thereby possibly compensating the initial loss of utility due to perturbation.

From Early Works. *"For reasons of modesty, fear of being thought bigoted, or merely a reluctance to confide secrets to strangers, many individuals attempt to evade certain questions put to them by interviewers."* states Warner in [Warner65] to motivate the seminal *randomized response* technique. To overcome the natural resistance of an individual in delivering his personal sensitive information to an unknown interviewer, Warner imagines letting the individual randomly perturb his data before giving it to the interviewer. The interviewer finally reconstructs the original data distribution based on the complete set of perturbed responses collected, and on the perturbation distribution. The gain in privacy resulting from the perturbation is expected to favor truthful responses to the survey, thus reducing the statistical bias due to evasive answers and increasing the utility of the collected dataset.

Let us describe more precisely the randomized response technique. The interviewer asks Boolean questions to the individual (e.g., "Do you use illegal drugs frequently?"). The individual simply answers truthfully with probability p and lies with probability $1-p$. Let π denote the true proportion of "Yes" answers in the population. For each individual the interviewer receives a "Yes" with the following probability: $\texttt{Pr}(\text{"Yes"}) = \pi \cdot p + (1-\pi) \cdot (1-p)$. Hence:

$$\pi = \frac{\texttt{Pr}(\text{"Yes"}) - (1-p)}{2p-1}$$

Assume that the interviewer received m "Yes" answers from n respondents; he can estimate $\texttt{Pr}(\text{"Yes"})$ by m/n. Let $\hat{\pi}$ denote the estimator for π. Then:

$$\hat{\pi} = \frac{m/n - (1-p)}{2p-1}$$

To Modern Approaches. With modern *local perturbation* approaches to privacy-preserving data publishing, e.g., [Evfimievski03, Mishra06, Agrawal05b, Agrawal05a, Agrawal09], the interviewer is replaced by the untrusted central server representing the publisher, and the respondents by clients representing the individuals. A client has personal data and perturbs it independently when participating in survey(s) proposed by the publisher.

The foundational work by Evfimievski & al [Evfimievski03] considers an adversarial prior knowledge that consists of an arbitrary probability distribution on arbitrary predicates over unitary data (e.g., equalities, inequalities, subsets). Following implicitly the uninformative principle, they propose a privacy model that strictly limits

the distance between the prior and posterior adversarial probabilities to a predefined threshold. A central result of this work is the *amplification* property. It is a sufficient condition - on the output distribution of the randomized perturbation algorithm used - for enforcing the proposed privacy model. Informally speaking, the amplification property states that a sanitized data is almost equally likely to result from the perturbation of any initial data by the underlying algorithm. The amplification property preserves the individuals' privacy and at the same time allows useful statistical analysis based on the perturbed data and the perturbation distribution. It appears retrospectively that the randomized response technique [Warner65] satisfies the amplification property [Chen09a]. The subsequent work by Mishra & Sandler [Mishra06] assumes similar adversarial and privacy models as [Evfimievski03] and concentrates on increasing the number of data per user to be published. The FRAPP framework [Agrawal05b, Agrawal09] also builds on [Evfimievski03] but focuses on studying the utility aspects of amplification-based local perturbation techniques through the process of reconstructing the distribution of the initial data. The authors express the expectation of the *perturbed* distribution by the result of a multiplication between the *perturbation matrix* - where the cell (i, j) contains the probability of perturbing the i^{th} value to the j^{th} value - and *the vector containing the initial data distribution* (i.e., the true number of occurrences of each initial data). They then derive the conditions that the perturbation matrix must satisfy in order to maximize utility and still enforce the amplification property. Furthermore, they propose to randomize the perturbation probabilities to obtain stronger privacy guarantees (at the cost of a slight loss in utility).

2.3 Cryptographic Tools

Cryptography fundamentally aims at guaranteeing the *security* of communications over insecure channels (e.g., telephone lines, computer networks). The objective of this thesis is the adaptation of *centralized publishing* approaches (see Section 2.2) to a decentralized and adversarial context. Secure communications will be necessary. This section aims at presenting the background knowledge necessary for understanding the cryptographic tools used in this thesis for guaranteeing *data confidentiality*, and *data integrity and authentication*. Informally speaking, the data confidentiality requirement states that Alice - the sender - and Bob - the receiver - should be able to exchange *encrypted (or obfuscated) messages* over an insecure communication channel such that Charlie - the adversary - spying on the channel is unable to deduce significant information about the corresponding *plaintext (or raw) messages*. The *data integrity and authentication* requirement states that Bob should be able to check that messages

really come from Alice: in other words, they were neither *forged* nor *tampered* by Charlie. Note that data integrity can be checked without checking data authentication while the reverse makes no sense. To enforce data confidentiality and data authentication, Alice, Bob, or both, must know a piece of information that Charlie does not know. Such piece of information is called a *cryptographic key* (or *key* for short). *Secret key* (or *symmetric key*) cryptographic schemes assume that both Alice and Bob have the same keys, contrary to *public key* (or *asymmetric key*) cryptographic schemes.

Our presentation follows the usual two stages of cryptographic schemes proposals: first, we define the properties that we expect from the cryptographic scheme under study (definition stage), and second, we sketch cryptographic constructions that guarantee these properties (construction stage). In the other chapters of the thesis, we will refer to the (generic) expected cryptographic properties rather than to any (specific) cryptographic scheme in charge of guaranteeing them. However, we give here constructions based on (1) the current standard symmetric-key *encryption algorithm*, namely the *Advanced Encryption Standard*[13] (also called *AES* for short, and used for guaranteeing data confidentiality as well as data authentication) [NIST01] and (2) the current standard cryptographic hash algorithm, namely the *Secure Hash Algorithm* (also called *SHA* for short, and used for guaranteeing either data authentication or data integrity alone) [NIST08b]. Although we could theoretically use any cryptographic algorithm, we favor AES and SHA for two reasons: first, they are both current standards - therefore they are widely used in practice, and second they are often implemented in secure portable tokens by a highly efficient *hardware circuit* . We consequently present cryptographic constructions - that fulfill the expected properties - based on these two algorithms.

This section aims at providing the background knowledge necessary for understanding the cryptographic tools used in this thesis. Hence, we favor intuitions rather than formal aspects. For further details, we refer the reader to the authoritative textbooks and surveys [Stinson05, Goldreich01, Goldreich04, Goldreich05].

This section starts by presenting how to achieve data confidentiality: we describe the *semantic security* property that we expect from *encryption schemes*, and the way to guarantee it based on AES. Then, we focus on data integrity and data authentication following the same definition and construction stages.

[13]The Advanced Encryption Standard was standardized by the US National Institute of Standards and Technology (NIST) in 2001.

2.3.1 Data Confidentiality

Data confidentiality is guaranteed by encrypting raw data at the sender, transmitting it encrypted on the communication channel, and decrypting it at the receiver. Assuming that both the sender and the receiver have the proper cryptographic key, an encryption scheme is basically made of (1) an encryption scheme that inputs the key and the plaintext data, and outputs the corresponding encrypted (or obfuscated) data, and (2) a decryption scheme that inputs the key and the encrypted data, and outputs the corresponding decrypted plaintext data. In the following, $\mathtt{E}$ denotes the encryption scheme, $\mathtt{E}^{-1}$ the decryption scheme, and κ the symmetric key. The encryption scheme and its corresponding decryption scheme are such that $\mathtt{E}_\kappa^{-1}(\mathtt{E}_\kappa(x)) = x$, where $x \in \{0,1\}^*$ is plaintext data (e.g., a database record).

Expected Property

In our context, during the decentralized execution of a given privacy algorithm, tokens will exchange multiple encrypted messages. Consequently the encryption scheme used must guarantee the *semantic security with multiple messages* property. Informally speaking, this property states that the adversary should not be able to learn, from the set of encrypted messages sent, *significantly more than the lengths of their corresponding plaintext messages*. We can rephrase this property following the *simulation paradigm*: any information that can be obtained (feasibly) from an encrypted bitstring can be simulated from a random bitstring.

Semantic security is related to the notion of *computational indistinguishability* of encryptions. Let us sketch it intuitively in the *single message* setting (it can be straightforwardly extended to the *multiple messages* setting [Goldreich04]). Computational indistinguishability of encryptions can be intuitively understood as follows: a *computationally-bounded* adversary (e.g., usually considered to have probabilistic polynomial-time computational resources) must not be able to draw a significant distinction between the encryptions of two distinct plaintexts of the same size. In other words, from the adversarial viewpoint, a given encrypted message is likely to be the result of encrypting any plaintext message of the expected size. More generally, two probability ensembles[14] $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, ranging over bitstrings of size $n \in \mathbb{N}$, are computationally indistinguishable, denoted $\overset{c}{\equiv}$, if, given a sample from one distribution in a probability ensemble, the adversary cannot decide from which probability ensemble the sample comes. For example, let $\{X_n\}_{n\in\mathbb{N}}$ represent the probability ensemble representing the output distributions that correspond to the encryption of the plaintext message x and $\{Y_n\}_{n\in\mathbb{N}}$ to the encryption of y (both x and y having

[14] Loosely speaking, a probability ensemble is a set of probability distributions.

the same size). $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable, denoted $\{X_n\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{Y_n\}_{n\in\mathbb{N}}$, if an adversary, obtaining, e.g., $e = \mathtt{E}_\kappa(x)$, cannot decide whether e is the result of encrypting x or y. Semantic security and computational indistinguishability are equivalent [Goldreich04].

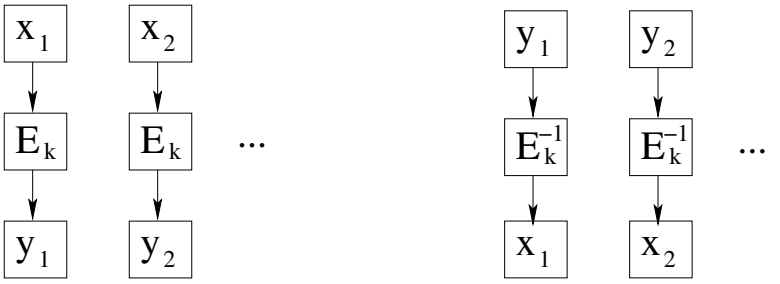
In our specific setting, we require the semantic security property to hold against *chosen-plaintext attacks* where the adversary is able to choose a plaintext message and to obtain its encryption (even though in the protocols that this thesis proposes, the adversary will not be able to obtain the encryption corresponding to a fully arbitrary plaintext). Standard modes of operation (described below) provide such guarantees. Note that there also exist non-standard modes, e.g., [Rogaway03], that provide semantic security against the strongest form of attack, called the *ciphertext-chosen attack*, where the adversary arbitrarily chooses a ciphertext and obtains the corresponding plaintext.

The AES Block Cipher

The AES algorithm [NIST01], standardized in 2001 by the US National Institute of Standards and Technology[15] (NIST), is a *block cipher*: it inputs a *block* of data (i.e., a bitstring of a fixed size - e.g., 128 bits for AES) and outputs a block of the same size (input data that do not have a size multiple of the block size are *padded*). AES is secure against all known attacks [Stinson05]. However, the security of a block cipher is insufficient for guaranteeing the security of the complete encryption scheme. Indeed, in practice, input data seldom holds in a single block: guaranteeing independently the security of each single block is far from implying that the security of the input data as a whole is guaranteed. *Operation modes* [Dworkin01, Dworkin07b] define how an arbitrarily sized input data is encrypted based on a block cipher. In our context, we fix the block cipher to AES; the encryption and decryption schemes thus result from the choice of an operation mode appropriate for guaranteeing the desired property against adversarial attacks (described below). Any operation mode can be chosen provided that it does not thwart the desired property.

AES is a *substitution-permutation network* block cipher (*SPN* for short). This type of cipher is made of a fixed number of iterations over the input block, where an iteration consists mainly in a substitution operation (deterministically substituting bytes in the block according to a pre-defined table), and in a deterministic permutation. A key-dependent operation makes the output of the SPN depend on the key. Substitution-permutation networks are thus fully deterministic.

[15] http://www.nist.gov/



(a) ECB mode (encrypt) (b) ECB mode (decrypt)

Figure 2.3: ECB Operation Mode

The AES cipher has a block length of 128 bits. The number of iterations that it performs depends on the size of the key: with a 128-bit key, AES performs 10 iterations, with a 192-bit key, 12 iterations, and with a 256-bit key, 14 iterations. Each iteration performs the same following sequence of operation (the last one differs a bit from the others): (1) let s be the current state of the input block (at the first iteration, s is set to the block), (2) **x-or** s with a bitstring depending on the key (where the verb *x-or* denotes the action of computing the bitwise *exclusive OR*), (3) **substitute** the bytes of s, and (4) **permute** the bytes of s. After the last iteration, s has reached its final, completely encrypted, state; AES outputs it and terminates.

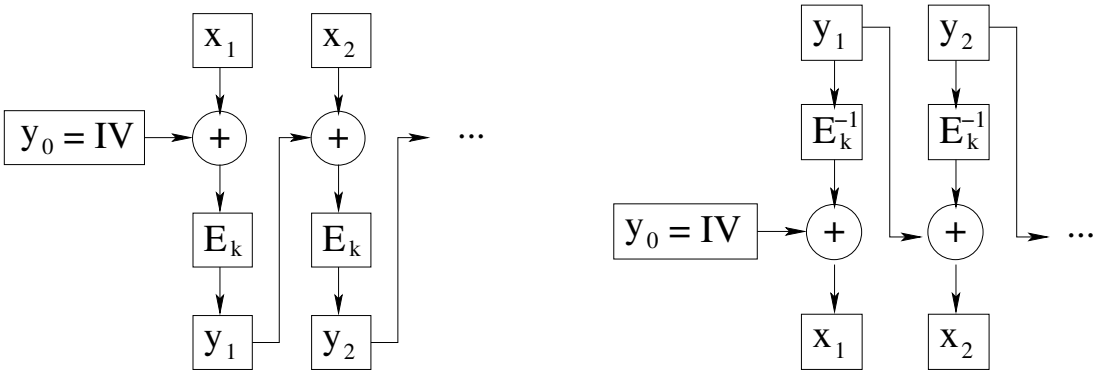
Operation Modes

The NIST has standardized various operation modes [Dworkin01, Dworkin07b, Dworkin07a, Dworkin10] for secret-key block ciphers. Let x_1, x_2, ... denote the sequence of input data blocks, and y_1, y_2, ... the sequence of encrypted data blocks.

Electronic Codebook Mode (ECB). The ECB mode corresponds to the straightforward use of a block cipher. As depicted in Figure 2.3(a), each block of input data is encrypted independently and based on the same key by the block cipher: $y_i = \mathtt{E}_\kappa(x_i)$ for all x_i. The decryption (Figure 2.3(b)) is the reverse process. The ECB mode provides poor security guarantees because two identical input blocks yield identical encrypted blocks. Figure 2.5(b) illustrates this based on the encryption of an image[16] by the ECB mode.

Cipher Block Chaining Mode (CBC). The underlying principle of the CBC mode [Dworkin01, Dworkin10] is to chain the encrypted blocks together by x-oring each input block x_i with the block previously encrypted y_{i-1} (see Figure 2.4(a)). The

[16]The initial image was drawn by Larry Ewing lewing@isc.tamu.edu based on The GIMP drawing program.

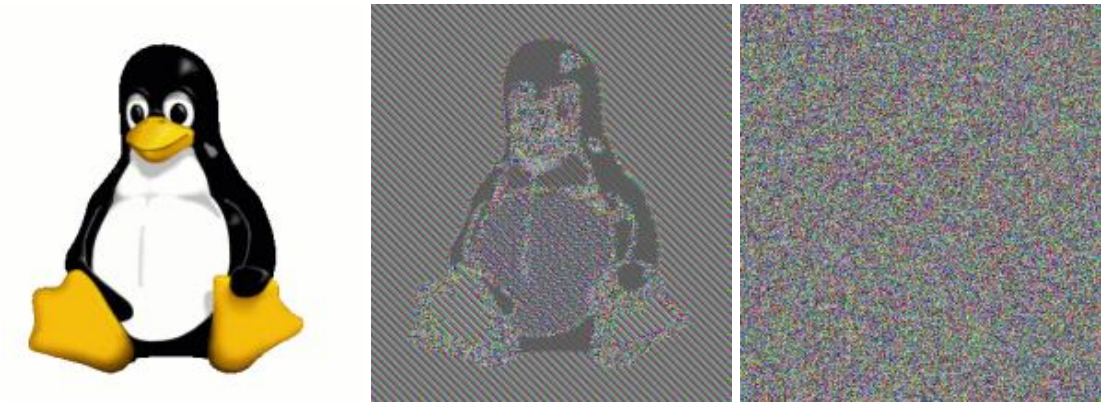


(a) CBC mode (encrypt) (b) CBC mode (decrypt)

Figure 2.4: CBC Operation Mode

initial input block is x-ored with an *initialization vector* (IV for short). The initialization vector for CBC must be an unpredictable bitstring of the same size of a block (e.g., a number used once, also called a *nonce*). The encryption in CBC mode follows: $y_i = \mathtt{E}_\kappa(x_i \oplus y_{i-1})$ for all x_i, where $y_0 = IV$. The decryption is the reverse process (the initialization vector, a single block, can be transmitted to the receiver, e.g., encrypted by the underlying block cipher). The CBC mode achieves the desired property stated above: semantic security with multiple messages against chosen plaintext attacks. Figure 2.5(c) illustrates this gain in security with respect to the ECB mode.

Other Operation Modes. The *output feedback* mode (*OFB* for short) [Dworkin01] generates a sequence of blocks (also called a *keystream*) by encrypting recursively the initialization vector. Let z_i be a keystream block: $z_i = \mathtt{E}_\kappa(z_{i-1})$, where $z_0 = IV$. Each keystream block is then x-ored with its corresponding input data block to produce an encrypted block: $y_i = z_i \oplus x_i$. The *cipher feedback* mode (*CFB* for short) [Dworkin01] differs from the OFB mode in the keystream generation: $z_i = \mathtt{E}_\kappa(y_{i-1})$. The *counter* mode (*CTR* for short) [Dworkin01] also differs from the OFB mode in that it uses a counter to generate the keystream rather than an initialization vector. As indicated by its name, the *counter with cipher-block-chaining* mode (*CCM* for short) [Dworkin07b] is based on the CBC and the CTR modes, and achieves simultaneously encryption (through the CTR mode) and data authentication (through the CBC mode). We will see below how CBC mode is used to guarantee data integrity and authentication. All these modes satisfy the desired property of semantic security with multiple messages against chosen-plaintext attacks.



(a) Initial (plaintext) (b) Encrypted (ECB Mode) (c) Encrypted (Other than the ECB Mode)

Figure 2.5: Operation Modes Illustrated[16]

2.3.2 Data Integrity and Data Authentication

Expected Properties

Properties Related to Data Integrity. An encrypted message sent by Alice to Bob over an insecure channel may be tampered. Bob must be able to ensure that the plaintext message is precisely the one that was encrypted and sent by Alice. A common approach consists in sending, with the encrypted message, a *compressed version*, also called a *message digest*, of the plaintext message. Provided that the scheme producing the digest is a *cryptographic hash function*, then any tampering of the encrypted message is detected (with high probability) when Bob decrypts the encrypted message, computes the digest of the plaintext, and compares the computed digest with the received digest. Let us now identify the properties that cryptographic hash functions should provide.

First, it is obvious from the above example that the adversary must not be able to compute a plaintext message based on the message digest. A cryptographic hash function must consequently be *one-way*, i.e., a function *easy to compute but hard to invert*. Let $\mathtt{H} : \{0,1\}^* \to \{0,1\}^n$ be the targeted cryptographic hash function, where $n \in \mathbb{N}$ is usually small (e.g., 160 bits). Given a plaintext message x, $\mathtt{H}(x)$ must be efficiently computable, while given $\mathtt{H}$ and $\mathtt{H}(x)$, finding x must be infeasible. Such a property is called the *pre-image resistance*.

Second, assume a different setting where the adversary knows the cleartext message but cannot tamper the associated digest (it is already known by Bob). An example of this setting could be the following. Alice wants to share some non-confidential data

with a large dynamic set of recipients. To this end, she stores the data in the cloud and shares the location with the recipients. In order to guarantee data integrity, Alice sends to the data recipients a digest of the data (computed beforehand and stored in a secure location - e.g., her tamper-resistant smart-card), so that they can compare it to the actual data digest. The only attack possible is to replace the data stored in the insecure location by another bitstring that has the same digest. The resistance to such attack is called the *second pre-image resistance*; it states that given $\mathtt{H}$, $\mathtt{H}(x)$, and x, finding x' such that $x' \neq x$ and $\mathtt{H}(x') = \mathtt{H}(x)$ must be infeasible.

More generally, finding a *collision* must be hard: given $\mathtt{H}$, finding x and x' such that $x' \neq x$ and $\mathtt{H}(x') = \mathtt{H}(x)$ must be infeasible. This is the *collision resistance* property. Intuitively, we feel that a collision-resistant function is also pre-image resistant and second pre-image resistant (see [Stinson05] for a formal analysis).

These properties become even more pregnant when we know that a cryptographic hash function *compresses* input data of arbitrary size to a reduced, fixed size, digest (e.g., 160 bits).

Data Integrity and Data Authentication Together. In addition to guaranteeing data integrity, *message authentication codes* (*MAC* for short) schemes guarantee data authentication. Assume that Bob receives a - not secret - plaintext message and its digest. He must be able to assert that this is really Alice that sent the message, and not an adversary having generated the plaintext message and computed its digest. MAC schemes are sometimes called *secret key signature schemes*. To avoid confusion, we underline their difference with *public key signature schemes*. In secret-key signature schemes, both the party that produces the signature (e.g., Alice the sender) and the party that verifies the signature (e.g., Bob the receiver) know the same secret key. As a result, (1) both the signer and the verifier are able to (secret key) sign a message and (2) nobody else than the sender and the verifier is able to verify the integrity and authenticity of a message based on its (secret key) signature. On the contrary, public key signature schemes use two distinct keys: a key for signing - which is *private* so only known by the signer - and a key for verifying - which is *public*. As a result, (1) the signer *only* is able to (public key) sign a message, and (2) anybody can verify the (public key) signature based on the public key. In the rest of the thesis, when mentioning signature schemes we implicitly designate *secret-key* signature schemes.

Let us now sketch cryptographic constructions for enforcing the desired properties.

Data Integrity: the Secure Hash Algorithm

The *Secure Hash Algorithm* (*SHA* for short) [NIST08b] is a standard from the NIST specifying a family of cryptographic hash functions. We sketch it below.

Iterated hash functions input a bitstring of an arbitrary size and compress it through recursive calls to a compression procedure in order to output a fixed size bitstring. They are generally built as follows. The input bitstring x is first preprocessed (e.g., padded and divided into blocks of fixed size) to yield the bitstring $y = y_1||y_2||\dots||y_r$, where $||$ denotes the bitwise concatenation and y_i a fixed size block. The compression process is now ready to start. A compression procedure, say `compress`, is in charge of compressing a bitstring of a *fixed size*, to a smaller bitstring (of a fixed size too). The `compress` procedure inputs each block y_i concatenated to the output of its previous call - denoted z_{i-1} - and outputs a fixed size block - denoted z_i. The compression process can thus be recursively defined by: $z_i = \texttt{compress}(y_i||z_{i-1})$, where z_0 is an initialization vector. The final output block z_r is optionally postprocessed.

The *Secure Hash Algorithm* (*SHA* for short) [NIST08b] is an iterated hash function. At the time of writing this thesis, three versions of SHA have been standardized (i.e., SHA-0, SHA-1, and SHA-2), and a new version, SHA-3, is planned for 2012[17]. SHA-1 is currently the most used SHA version. It inputs a message of at most 2^{64} bits. The message is then preprocessed by being padded to a size multiple of 512, and divided into blocks of 512 bits. The recursive compression procedure, `compress`, is now ready to start. The initialization vector is made of a constant 160-bit number. At each iteration, the `compress` procedure inputs a bitstring of 160 + 512 bits and outputs a 160-bit bitstring. The internals of `compress` consist in bitwise operations on the input string (extract, rotate, and, or, xor). We refer the interested reader to [Stinson05, NIST08b] for details (explaining them does not present much interest here).

Data Integrity and Authentication: Message Authentication Codes

There exist two main practical ways for producing *message authentication codes* (*MAC* for short): the *keyed-hash* approach - which consists in incorporating a secret key in a cryptographic hash function - and the *block cipher mode of operation* approach which is primarily based on encryption modes of operation.

The Keyed-Hash Message Authentication Code. Including a key as part of the message to be hashed by a cryptographic hash function is a natural approach to guaranteeing both data integrity (the role of the hash function) and data authentication (the role of the key). Doing this without thwarting the key secrecy is however not trivial [Stinson05]. The Keyed-Hash Message Authentication Code [NIST08a] (*HMAC* for short) is the standard for this approach. The HMAC standard states that the HMAC's hash function can be any cryptographic hash function approved by the NIST. When using SHA-1 with a 512-bit key, the HMAC is simply computed by: $\texttt{HMAC}_\kappa(x) =$

[17] http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

SHA-1 ($(\kappa \oplus opad)$ || SHA-1 ($(\kappa \oplus ipad)$ || x)), where x is the input message, κ is the 512-bit secret key, and *ipad* and *opad* are 512-bit pads ($ipad = (36 \dots 36)_{16}$ means *inner pad*, and $opad = (5C \dots 5C)_{16}$ means *outer pad*).

The CMAC Mode for Authentication. Recall that the CBC mode of operation chains the complete sequence of input blocks: in CBC mode, the i^{th} output block is defined by: $y_i = \mathtt{E}_\kappa(y_{i-1}||x_i)$, where x_i is the i^{th} input block, κ is the secret key, and y_0 is the initialization vector. As a result, the bits of the last output block depends on the bits of the complete input message. The principle of the CMAC scheme [Dworkin05] is based on this observation and essentially consists in computing the MAC of a message x based on the last block resulting from x's CBC encryption.

2.4 Related Approaches

To the best of our knowledge, though a few approaches consider a context similar to this thesis, no work has tackled simultaneously the **Decentralization**, **Genericity**, and **Generality** objectives for centralized publishing algorithms so far.

The *Secure Multi-party Computation* (*SMC* for short) approach has investigated the adaptation of privacy-preserving data publishing algorithms to a decentralized setting. The SMC problem consists in allowing several parties (e.g., hospitals) to jointly compute a function (e.g., a sanitization algorithm) such that no party learns anything about the other party's input (e.g., their local databases) beyond the function's result (e.g., the sanitized union, or join, of the local databases). The *generic* SMC approach aims at providing generic constructs for translating *any* centralized function to a secure decentralized protocol. The resulting protocols however exhibit a prohibitive cost for practical settings, missing the **Generality** objective. This has motivated the investigation of *specific* SMC protocols, each dedicated to achieving - efficiently - a given precise decentralized function. However, since they are specific, they obviously miss the **Genericity** objective (in addition to the **Generality** objective).

Token-based implementations of secure multi-party computation protocols are receiving an increasing attention. However, to the best of our knowledge, this approach has not considered privacy-preserving data publishing algorithms so far.

Finally, the *anonymity-preserving data collection* approach does not consider that the identifying information of an individual appears in her data, but was disclosed (e.g., given, inferred) during the collection process. Consequently, anonymity-preserving data collection techniques do not *transform* data but *unlink* it from the precise respondent to which it is related. They address a context different from the one addressed by privacy-preserving data publishing approaches.

This section starts by introducing the generic secure multi-party computation approach, explaining why it is inadequate for adapting centralized data publishing algorithms to a decentralized context. Second, we describe the existing specific secure multi-party computation approaches dedicated to centralized publishing algorithms and position them with respect to the three objectives mentioned above. Finally, we overview the related token-based and anonymity-preserving data collection approaches.

2.4.1 The Generic Secure Multi-Party Computation Approach

Initiated by Yao [Yao82, Yao86] and Goldreich & al [Goldreich87], the study of *generic* SMC protocols aims at providing constructs for translating *any* centralized function to a secure decentralized protocol.

Generic SMC approaches roughly follow the same pattern. To start, the centralized function is expressed as a *combinatorial circuit*. If the circuit were to be executed centrally, each bit of the input would be entered into an input wire, and then be propagated through the gates. In the SMC context, the circuit must be decentralized, and *garbled* such that the bits of each party's input be obfuscated to the other parties. Therefore subsequently, gates of the circuit are translated to a secure protocol to be executed by the parties that hold the gate's inputs. Finally, the complete garbled circuit is executed.

The above sketch is sufficient for observing that the resulting cost depends on the number of gates in the circuit, which in turn depends exponentially on the size of the input data and on the complexity of the initial function. For example, a naive combinatorial circuit for multiplying two integers has quadratic size (in the number of bits of the integers input). Despite their unquestionable theoretical interest, these generic approaches cannot be directly used in practical privacy-preserving data publishing scenarios where inputs are large and sanitization algorithms complex. They achieve the **Genericity** objective but not the **Generality** objective.

2.4.2 Secure Multi-Party Computation Approaches Dedicated to Centralized Publishing Algorithms

Secure multi-party computation approaches dedicated to centralized publishing algorithms primarily consider that the (virtual) dataset to sanitize is distributed over a set of mutually distrustful parties. Each party may hold either a *horizontal partition* of the dataset (i.e., a subset of records) or a *vertical partition* (i.e., a subset of attributes) of the dataset. These two ways of considering the data hosted by the distributed parties (also called, the *local* data) has led to two orthogonal lines of work. We present both

lines below, focusing especially on the works most closely related to this thesis, and overviewing the others. Note that a related approach is described in [Trombetta11] where the impact of inserting a record into a central encrypted database on its k-anonymous view is studied. However, this approach focuses on the updates and not on the initial production of the k-anonymous view.

Horizontally Partitioned Data

Privacy-Enhancing k-Anonymization of Customer Data [Zhong05]. The authors of [Zhong05] aim at enforcing the k-ANONYMITY privacy model without any central server. The setting considered consists of individuals that hold private data, and a miner interested in this data. The individuals and the miner are all assumed *honest-but-curious* in that they try to infer all that they can but strictly abide the protocol. The authors make a judicious usage of specific cryptographic schemes in order to permit the individuals and the miner to jointly compute a k-anonymous release of the union of the individuals' data. The authors propose two protocols, proven to fulfill the strong computational indistinguishability requirement (see Section 2.3) modulo some information voluntarily disclosed.

First, Zhong & al propose to extract the part of the virtual dataset that is *naturally* k-anonymous, i.e., only the records whose quasi-identifiers are indistinguishable from at least $k-1$ other records *without any transformation*. The protocol consists in a single phase. Individuals participate to the protocol by sending to the miner their quasi-identifier value in the clear, their sensitive value encrypted, and a *share* of the corresponding decryption key. Indeed, each individual defines her own encryption/decryption keys such that (1) the encryption key depends on the quasi-identifier value and (2) the *full* decryption key can be recovered only with at least k distinct shares. As a result, the miner will be able to decrypt the sensitive value only when he will have gathered at least k distinct shares of a key corresponding to the same quasi-identifier value. In other words, he will be able to decrypt only the sensitive values of records that are naturally k-anonymous.

Second, since only few records may be naturally k-anonymous, the authors tackle the sanitization of the entire table by adapting the centralized publishing algorithm proposed in [Meyerson04] which is based on suppression[18]. The resulting protocol consists of three phases. First, individuals send to the miner their quasi-identifier value encrypted attribute per attribute. The encryption scheme is such that the miner, with the help of the individuals, is able to compute the *distance* between each pair of

[18]Suppression-based algorithms are usually simpler than generalization-based algorithms because suppression is similar to a maximal generalization so does not consider the intermediate nodes of the generalization hierarchy.

quasi-identifier, where the distance is defined as the number of attributes on which their respective corresponding values are different. Second, the miner computes a set of equivalence classes by executing the centralized publishing algorithm [Meyerson04] on the encrypted quasi-identifiers based on the distances computed previously. Finally, each individual downloads the class in which her quasi-identifier appears, and, based on a specific cryptographic scheme, degrades her own quasi-identifier according to the class and sends it to the miner with her sensitive value.

The approach meets the **Decentralization** objective but fulfills neither the **Generality** objective nor the **Genericity** objective. Indeed, the individuals that connected during the first phase are expected to be connected during the third phase. This is a strong assumption in our context because individuals primarily host their data on autonomous devices possibly highly disconnected (see Chapter 6); the **Generality** objective is not reached. Obviously, the **Genericity** objective is not fulfilled either because the proposed protocols are strongly tied to the k-ANONYMITY model (for both protocols) and to the algorithm proposed in [Meyerson04] (for the second protocol). Moreover, the honest-but-curious attack model that the miner is expected to follow is also a strong assumption; extending the approach to a stronger attack model does not appear straightforward (e.g., precluding the tampering of the equivalence classes by the miner).

Distributed Privacy Preserving Data Collection [Xue11]. The approach proposed in [Xue11] is similar to [Zhong05]; the difference lies in that they use generalization instead of suppression. The approach suffers from similar shortcomings.

k-Anonymous Data Collection [Zhong09]. The work proposed in [Zhong09] tackles a context similar to [Zhong05]: a set of individuals hold their own records and a miner wants to obtain the k-anonymous release of the union of their records. In addition, the authors assume that the communication channel *identifies* the individuals participating in the protocol. This assumption makes the second protocol proposed in [Zhong05] inapplicable because the link between each individual and her sensitive data would be trivially disclosed during the third phase. The underlying security model is also computational indistinguishability. Note that a certain participant, called the *Data Collection Helper* (*DCH* for short), endorses a central role, sketched below, requiring that it does not collude with the miner.

The basic version of the protocol copes with a miner and a DCH that follow the *honest-but-curious* attack model. First, the quasi-identifier and sensitive values are collected by the miner, encrypted separately based on the public keys of both the miner and the DCH. The encrypted tuples are then shuffled (by the miner and the DCH), made k-anonymous (by the DCH that, without decrypting the records, suppresses *completely* the quasi-identifiers that appear less than k times), and finally decrypted

(by the DCH and the miner). This protocol is then extended to a stronger attack model that considers active attacks from the miner and the DCH (but still requires that they do not collude together).

The protocols proposed in [Zhong09] relax the high availability requirement formulated in [Zhong05] concerning the complete set of individuals, but still requires full availability of the participant endorsing the role of DCH. The **Generality** objective is consequently not reached. The **Genericity** objective is clearly not fulfilled because the protocols are designed to enforce the k-ANONYMITY model by suppression. Moreover, the interest of the approach is unclear: the proposed protocols degrade data more than their centralized counterparts, thus possibly losing the benefits expected from centralized publishing algorithms, and consequently questioning its interest with respect to simpler local perturbation approaches.

Privacy-Preserving Data Publishing for Horizontally Partitioned Databases [Jurczyk08a, Jurczyk08b, Jurczyk09]. In [Jurczyk08a, Jurczyk08b, Jurczyk09], the authors propose an adaptation of the MONDRIAN generalization-based algorithm [LeFevre06] (see Section 2.2) for producing a k-anonymous release of the union of the datasets. The underlying security model is computational indistinguishability modulo a controlled amount of information voluntarily disclosed.

The set of participants is mapped beforehand to a *circular* overlay network. They execute a decentralized adaptation of the MONDRIAN algorithm by using a set of secure multi-party computation protocols as building blocks (i.e., secure sum, secure min/max, and secure median protocols). The execution sequence is under the control of a single leading site and assumes that participants are fully available.

The approach does not meet the **Genericity** objective (it is strongly tied to the k-ANONYMITY privacy model and the MONDRIAN algorithm) nor the **Generality** objective (it assumes that participants are highly available).

Centralized and Distributed Anonymization for High-Dimensional Healthcare Data [Mohammed10]. The approach proposed in [Mohammed10] differ from [Jurczyk08a, Jurczyk08b, Jurczyk09] in the underlying privacy model and algorithm, but both are similar. With respect to our context, [Mohammed10] consequently suffers from the same shortcomings.

Vertically Partitioned Data

A Secure Distributed Framework for Achieving k-Anonymity [Jiang06]. In [Jiang06], the authors consider a database vertically partitioned over two sites. The partitions have attributes in common: a subset of the quasi-identifier, and a global unique identifier. The goal of [Jiang06] is to k-anonymize the dataset resulting from

joining the two partitions on the global identifier, such that each party does not learn more about the other party's data than the k-anonymous release. The security model considered is computational indistinguishability in a honest-but-curious attack model.

The basic idea of the proposed protocol is to let the two parties locally generalize their data, and check whether, if joined, the resulting equivalence classes would contain at least k quasi-identifiers. The check is based on a secure set intersection protocol. If the check fails, then local data is more generalized. The process continues recursively until a k-anonymous join is found.

The approach tackles a different context: the dataset is partitioned vertically, only two parties are involved in the protocol, and their availability is high. Therefore it is not possible to use it directly with our requirements.

Privacy-Preserving Data Mashup [Mohammed09]. The authors of [Mohammed09] consider a context similar to [Jiang06], in that data is vertically partitioned over several parties and there exist a global identifier common to all partitions, and propose an approach similar to [Jurczyk08a, Jurczyk08b, Jurczyk09], in that they adapt a centralized publishing algorithm (i.e., [Fung05]) to this decentralized context by using a set of secure multi-party computation as building blocks.

This approach is consequently disqualified for use in our context (vertical partitioning, high availability assumption, and specificity of the privacy model and algorithm used).

The Secure Multi-Party Computation Toolkit

Most of the above approaches suffer from similar lacks in terms of **Genericity**. A noticeable attempt to favor **Genericity** is the secure multi-party computation toolkit sketched in [Clifton02]. The toolkit aims at gathering a small set of secure multi-party computation protocols whose efficiency and security are demonstrated. These protocols can then be combined in order to implement a wide variety of distributed complex privacy-preserving data mining algorithms. However, they are based on the assumption, common to secure multi-party computation approaches, that participants are highly available. The **Generality** objective is not reached.

2.4.3 Miscellaneous

Secure Multi-Party Computation Approaches Based on Tokens

The interest in cryptographic protocols founded on tamper-resistant tokens is resurging. Indeed, many works revisit the traditional approaches and their practical and the-

oretical results based on the use of secure hardware. For example, [Katz07, Goyal10a, Goyal10b] revisit the traditional theoretical results of secure multi-party computation protocols by benefiting from the tangible trust provided by tokens; [Jarvinen10] benefit from tamper-resistant tokens to decrease the communication cost of the generic SMC approach by equipping parties with tamper-resistant tokens which are - locally - in charge of a part of the circuit evaluation; [Hazay08, Fischlin11] propose two-party secure set intersection protocols based on a physical exchange of (preloaded) trusted tokens. To the best of our knowledge, no token-based approach to privacy-preserving data publishing has been proposed so far.

Anonymity-Preserving Data Collection

The *anonymity-preserving data collection* approach (e.g., [Yang05, Brickell06, Ashrafi09a]) considers an untrusted data miner wishing to collect data from a set of mutually distrustful individuals. This approach assumes that the link between an individual and her data is disclosed through the communication process to the miner (e.g., identification of individuals that respond to the miner) and is not contained in the data - as is usually assumed by privacy-preserving data publishing approaches. Some works consider *anonymous communication channels* [Ren10] as an alternative solution (though less practical according to the authors) to the problem (e.g., [Yang05, Brickell06]); some others consider such channels as an element of response [Ashrafi09b].

Anonymity-preserving techniques consequently define privacy as being the *unlinkability* between an individual and her responses (provided that responses do not contain identifying information). Basically, anonymous collection techniques [Yang05, Brickell06] usually let individuals encrypt their data and jointly shuffle the encrypted dataset. When the shuffling is over, the individuals and the miner collaborate for decrypting the dataset. The security against passive and active attacks has been considered, as well as the resistance to collusions.

Since it considers that individuals' data do not contain any identifying information, the anonymity-preserving data collection approach is orthogonal to the privacy-preserving data publishing approach considered in this thesis.

Chapter 3

Problem Statement

Summary: *In this chapter, we state the approach suggested in this thesis for tackling the* ***Decentralized****,* ***Genericity****, and* ***Generality*** *objectives in the context of non-interactive privacy-preserving data publishing. We analyze the execution sequence of centralized publishing algorithms and extract the common phases that they follow (i.e., the* ***collection****,* ***construction****, and* ***sanitization*** *phases). Our approach is based on an emerging type of device called the* secure portable token*. Tokens are expected to provide* high trustworthiness *(primarily due to their tamper-resistance),* low availability *(they are under their owner's control), and* modest computing resources*. Adapting centralized data publishing algorithms and models to a token-based context requires the re-introduction of an untrusted central supporting server, namely* the publisher*, and the remodeling of the* ***collection****,* ***construction****, and* ***sanitization*** *phases such that the collection phase is delegated to the publisher. The publisher follows two well-known attack models: the* honest-but-curious *and the* weakly-malicious *models. The correctness and security properties of the execution sequence have to be guaranteed against these two types of adversaries.*

Contents

3.1 Common Structure of Centralized Publishing Algorithms

BEFORE studying the decentralized context considered in this thesis, we step back a bit and discuss the traditional execution sequence of centralized publishing algorithms.

Centralized publishing algorithms are traditionally run by a *trusted* publisher. They base their *sanitization action*, denoted `S`, on prior knowledge of the complete dataset, which helps them achieve a better privacy/utility tradeoff than local perturbation algorithms [Rastogi07]. Section 2.2.2 presented a sample of the major centralized publishing algorithms; we now extract the - *common but implicit* - structure that they follow, and fix the notations and terms commonly used in this thesis (Table 3.2 summarizes them). We will later shape our protocol following this common structure. We close this section by illustrating how two types of algorithms exhibiting substantial differences, i.e., the generalization-based privacy algorithms and the $\alpha\beta$-ALGORITHM, follow the common structure.

3.1.1 Collection, Construction, Sanitization Phases

Centralized publishing algorithms start by a **collection phase** during which the trusted publisher collects data from individuals. In this thesis, we focus on tabular data. The collected dataset, also called the *initial dataset* and denoted $\mathcal{D}$, is thus a table consisting of $n_{\mathcal{D}}$ raw *records*. A record is made of one or more attributes, and takes its value from the domain of all possible records, denoted *dom*, which is the cross-product of the attributes's domains. When, e.g., the dataset reaches a sufficient cardinality, the publisher stops the collection phase and computes an algorithm-dependent data structure based on the dataset: this is the **construction phase**. The constructed algorithm-dependent data structure is called *sanitization rules* (rules for short) and denoted $\mathcal{R}$. The rules and the initial dataset are inputs of the **sanitization phase** during which the publisher produces the *sanitized release*, denoted $\mathcal{V}$.

The use of sanitization rules is the main reason for centralizing data: they allow the underlying algorithm to tune its sanitization action `S` *according to the actual dataset* (concrete examples are presented in the next section). The nature of rules is highly algorithm-dependent (e.g., rules are tuples generated randomly in the $\alpha\beta$-ALGORITHM [Rastogi07], or equivalence classes in generalization-based algorithms [Sweeney02]). The *privacy parameters* allow the publisher to choose the desired level of privacy to be reached by the sanitized release (e.g., the parameters d and γ of the (d,γ)-PRIVACY

model, the parameter k of the k-ANONYMITY model). In the following, the terms *privacy algorithms* refer to centralized publishing algorithms.

3.1.2 Illustrations

Let us illustrate the abstract phases described above through two different concrete types of algorithms.

Generalization-Based Algorithms

Generalization-based algorithms are structured according to the **collection**, **construction**, and **sanitization** phases as follows. The publisher starts by forming the initial dataset $\mathcal{D}$ by collecting the raw records of individuals; this is the **collection phase**. During the **construction** phase, the underlying algorithm computes the equivalence classes based on a subset of the records's attributes (typically, the attributes in the quasi-identifier). Note that equivalence classes may additionally have to satisfy some constraints on other attributes (typically, the sensitive attributes). The sanitization rules $\mathcal{R}$ precisely consist of the equivalence classes. The **sanitization phase** then produces the sanitized release $\mathcal{V}$ by performing the sanitization action `S` on the initial dataset based on the rules. Here, `S` consists, for each record r, in associating the generalization-node of the class that generalizes r's corresponding quasi-identifier value to r's sensitive value. Finally, the sanitized release is published.

$\alpha\beta$-Algorithm

The $\alpha\beta$-ALGORITHM is structured according to the **collection**, **construction**, and **sanitization** phases as follows. During the **collection phase**, the publisher forms the initial dataset $\mathcal{D}$ by collecting the raw records of individuals (note that the $\alpha\beta$-ALGORITHM considers that each record is unique). The **construction phase** consists in generating the rules $\mathcal{R}$: they are made of a predefined number $n_{\mathcal{R}}$ of distinct records randomly sampled from the records's definition domain *dom* such that $\mathcal{D} \cap \mathcal{R} = \emptyset$ (the actual value of $n_{\mathcal{R}}$ depends on the privacy parameters α and β). During the **sanitization phase**, the publisher performs the $\alpha\beta$-ALGORITHM's sanitization action `S`: it inserts into the sanitized release $\mathcal{V}$ (1) each tuple from $\mathcal{D}$ with probability $\alpha + \beta$, and (2) all the tuples from $\mathcal{R}$. Finally, $\mathcal{V}$ is published with the privacy parameters α and β, and the domain of all possible records *dom*. The privacy guarantees enforced by the $\alpha\beta$-ALGORITHM lie in the fact that the probability of identifying a real record in the sanitized release is bounded and can be set as desired.

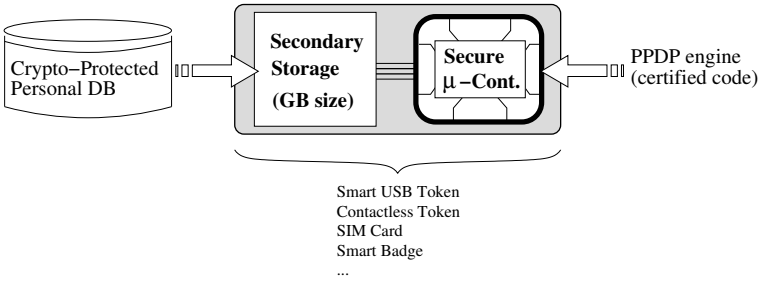


Figure 3.1: Smart Portable Tokens

3.2 When Secure Portable Tokens Empower Individuals

In this thesis's context, the records of each individual are primarily hosted by the individual's *secure portable token*. Whatever their form factor, secure portable tokens are usually composed of a *tamper-resistant* micro-controller connected by a bus to a gigabytes size external secondary storage area (see Figure 3.1). We describe below the properties that we expect a token to exhibit.

3.2.1 High Security Guarantees

A token provides a *trustworthy computing environment*. This property relies on the following security guarantees provided by a token:

- The microcontroller is tamper resistant, making hardware and side-channel attacks highly difficult;
- Software is certified according to the Common Criteria certification[19] making software attacks highly difficult;
- The embedded software can be auto-administered more easily than its multi-user central server counterpart thanks to its simplicity, removing the need for DBAs and therefore eliminating such insider attacks;
- Even the token's owner cannot directly access the data stored locally (she must authenticate, using a PIN code or a certificate, and only gets data according to her privileges);

The token's trustworthiness stems from the expected high *Cost/Benefit* ratio of an attack: tokens enforce the highest hardware and software security standards (prohibitive costs), and each of them hosts the data of a single individual (low benefits).

[19] http://www.commoncriteriaportal.org/

3.2.2 No Guarantee of Availability

A token provides *no guarantee of availability*: it is physically controlled by its owner who connects and disconnects it at will.

3.2.3 Modest Computing Resource

A token provides *modest computing resources*. Although the tamper resistance requirement restricts the general computing resources of the secure environment, it is common that dedicated hardware circuits handle cryptographic operations efficiently (e.g., dedicated AES and SHA hardware implementations). The secure environment also contains a small amount of persistent memory in charge of storing the code executed in the token and the *cryptographic keys* (also called *cryptographic material*). For the sake of simplicity, we assume that each token already contains its cryptographic material and privacy parameters before the PPDP protocol starts. Chapter 6 discusses practical ways of setting these pre-required data structures.

3.3 Revisiting Privacy-Preserving Data Publishing

Now that we have described the properties that this thesis expects from secure portable token, and extracted the common execution sequence of centralized publishing algorithms, we are ready to introduce our approach to privacy-preserving data publishing. Recall that we focus on the non-interactive setting, which consists of the local perturbation and centralized publishing families. As explained in the previous chapter (see Section 2.2.4), local perturbation algorithms are already fully decentralized, so fit our token-based approach *out-of-the-box*. On the contrary, performing centralized publishing algorithms in a decentralized context is challenging (the terms themselves are contradictory). The following consequently focuses on centralized publishing algorithms.

3.3.1 The Case for a Participating Supporting Server

A natural approach to tackle the problem could consist in designing a distributed protocol involving only tokens, without any central server. They would share their data together and construct jointly the sanitization rules, e.g., in a *peer-to-peer* fashion. A token is however an autonomous and highly disconnected device, that moreover remains under the control of its owner. Guaranteeing the availability of both the data and the results of intermediate computation given such highly volatile devices would incur a

Privacy Model	Privacy Algorithm
k-Anonymity	Generalization-based algorithms
ℓ-Diversity	Mondrian
idem	Bucketization
t-Closeness	SABRE
(d, γ)-Privacy	$\alpha\beta$-Algorithm
ϵ-Privacy	Mondrian

Table 3.1: A Sample of Supported Models and Algorithms

prohibitively high network cost (data transfers between tokens). Such an approach would fail to meet the **Generality** objective stated in the introduction.

A central supporting server is thus needed; we call it the *publisher*. Let us assume for the moment that its role is minimal: the publisher only serves as an *untrusted* storage area for both data and intermediate computations. In this approach, tokens would start by encrypting their data, then uploading it on the publisher. Once enough data would have been collected, tokens would construct the sanitization rules over the encrypted data: each connecting token would download a partition of the encrypted dataset, decrypt it, perform a small unitary task of a parallelized version of the chosen privacy algorithm, and upload the encrypted intermediate result (the token's modest computing resources and low availability preclude it to download and process the complete dataset at once). For example, we could imagine to parallelize the Mondrian generalization-based algorithm based on a recursive distributed sort algorithm over the encrypted dataset. However, this approach does not reach our objectives neither: although it *may* achieve the **Generality** objective (the actual cost would however depend on the internals of the parallelized algorithm), it clearly would not achieve the **Genericity** objective because each new algorithm adaptation would require to redesign - from scratch - a specific parallel protocol, and prove its correctness and security.

We see in the above, the emergence of a remodeling of the **collection**, **construction**, and **sanitization** phases. We claim that the key to achieving the **Generality** and **Genericity** objectives together is to *delegate the* ***construction*** *phase to the publisher* (possibly helped by tokens). Indeed, the **construction** phase is both hardly parallelizable (it cannot be performed independently on subsets of records but rather involves the complete dataset) and highly algorithm-dependent (its internals vary depending on the privacy algorithm performed). On the contrary, the **collection** and **sanitization** phases operate naturally on subsets of tuples rather than on the complete dataset at once (which makes them easily parallelizable) and can be smoothly abstracted from the underlying privacy algorithm (as will be shown along the thesis).

In order to delegate the **construction** phase to the publisher, tokens must disclose

sufficient data for allowing it to compute the sanitization rules based on the traditional centralized publishing algorithms. The resulting execution sequence consists thus in the following remodeling of the traditional phases. During the **collection** phase, each participating token sends to the publisher a *tuple*[20] made of its owner's record *partially* obfuscated such that the publisher can use it for constructing rules but cannot access the raw record. The tuples collected form the *obfuscated dataset*, denoted $\mathcal{D}_o$. During the **construction** phase the publisher computes the sanitization rules, denoted $\mathcal{R}$, based on the information sent in the clear during the **collection** phase. Finally, during the **sanitization phase**, tokens perform in parallel the sanitization action, on subsets of the collected dataset, and based on the sanitization rules. This approach[21] obviously assumes that the partial disclosure necessary for enabling the participation of the *untrusted* publisher does not thwart the privacy guarantees of the underlying model. Table 3.1 shows that this assumption is often satisfied in practice by listing a wide range of privacy models and algorithms supported by this approach (*how* they are supported will be clarified in Chapter 5). Moreover, the design of new privacy algorithms especially tailored to this approach is open to future work, and expected to enlarge the family of supported privacy models.

We describe precisely the resulting architecture below, and formalize the security and correctness properties to be asserted by the approach.

3.3.2 The Asymmetric Architecture

The architecture on which we base our approach consists on the one hand of a (large) set of trustworthy tokens hosting individuals' data, and on the other hand of an untrusted participating supporting server. We call it the Asymmetric *architecture* due to the asymmetry (trust, availability, resources) between the tokens and the publisher (as depicted in Figure 3.2). Indeed, being a traditional central server, the publisher does not suffer from specific resource constraints: it provides *standard computing resources* and exhibits a *standard availability*. Its role is to make up for the absence of availability

[20] For the writing clarity, we use the term *record* for designating a (plaintext) row of the dataset to be sanitized, and the term *tuple* for designating the data structure sent by a token to the publisher during the **collection** phase and which contains notably a (partially obfuscated) record.

[21] Some related works such as, e.g., [Zhong05], have tackled the problem of distributing a centralized publishing algorithm over a publisher and a set of participants. Although they tackle a different context (e.g., participants do not trust each other, the attack model is limited to honest-but-curious, the underlying algorithm is a specific generalization-based algorithm), it is interesting to note that they can be presented along the lines of the **collection**, **construction**, and **sanitization** phases, where the **construction** phase is delegated to the honest-but-curious publisher by disclosing it a controlled amount of information (e.g., the distance between the quasi-identifier values). For more details on related works, see Section 2.4.

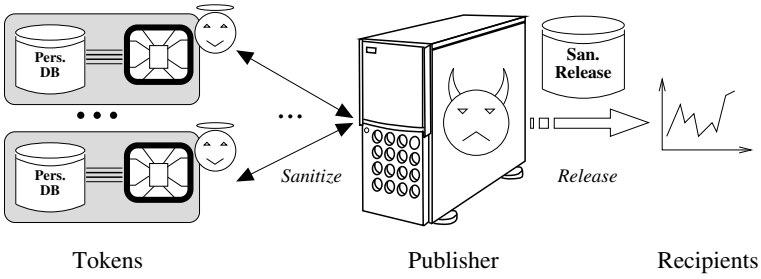


Figure 3.2: The ASYMMETRIC Architecture

guarantees provided by tokens: it is essentially in charge of supporting the construction phase of centralized publishing algorithms. However, this is not trivial since it is *not trustworthy*.

3.3.3 Attack Model

In this work, we consider that the publisher follows two well-known types of attack models.

Honest-but-Curious Adversary

The *honest-but-curious* adversary (also called *semi-honest* or *passive*) does not deviate from the protocol but infers anything that is feasible to infer about the collected tuples (see Section 2.3 for the notion of computational feasibility). Inferences aim at narrowing the link between subsets of data disclosed in the clear during the **collection phase** and their corresponding subsets of sanitized records returned in the clear during the **sanitization phase**. We will clearly describe this link and demonstrate its sensitivity in the following chapters. In practice, the honest-but-curious attack model fits well-established publishers (e.g., a governmental agency). We additionally use it as an appropriate baseline for laying the foundations of each suite of protocols.

Weakly-Malicious Adversary

Although the *weakly-malicious* adversary [Zhang05] also aims at drawing inferences, it differs from the previous attacker in that it may deviate from the protocol as long as (1) it is not convicted as an adversary by another party (i.e., a token), and (2) it is able to produce a correct result. In practice, the weakly-malicious attack model fits well a third-party publisher (e.g., a software-as-a-service company).

The scope of *active attacks* encompasses the data structures used by the protocol (i.e., the dataset and the sanitization rules), and the execution sequence itself. Hence, active attacks are threefold. First, the attacker may tamper the dataset; tokens must guarantee its integrity against the usual *forge*, *copy*, and *delete tampering actions* (tampering actions that modify a tuple are covered by the forge actions). A small set of safety properties checked by the tokens is sufficient for *guaranteeing the dataset's integrity* (e.g., assert the origin and integrity of each tuple based on its MAC). Second, since the **construction phase** is executed on the attacker's side, it can simply build unsafe sanitization rules (e.g., constructing equivalence classes that actually generalize less than k tuples whereas they are supposed to be k-anonymous). Checking the *consistency between the rules and the dataset* is thus crucial. Third and last, the attacker may disorganize the protocol execution sequence by performing it *partially* or in a *different order* (e.g., omiting to check a safety property). Tokens must also check the *integrity of the execution sequence*. We will thoroughly describe attacks and counter-measures in the following chapters.

As a result, by precluding any active attack, safety properties reduce active adversaries to launching passive attacks only, which have already been disabled when tackling the honest-but-curious attack model.

3.3.4 Correct and Secure Computation

Within our approach, the privacy algorithm is enforced through a distributed protocol between a set of (trusted) tokens and the (untrusted) publisher. Demonstrating that the protocol computes effectively the centralized publishing algorithm to enforce - *correctness* - and without any data leak - *security* - is thus crucial. We detail below the model based on which the protocol proposed in this thesis is shown to be correct and secure. The model is grounded in the strong and well-known *computational indistinguishability* notion [Goldreich05], which is the state-of-the-art standard for assessing the security of multi-party computation protocols. Stated informally, two distribution ensembles are computationally indistinguishable if no probabilistic polynomial-time algorithm is able to draw a significant difference between them (we refer the reader to Section 2.3 for details).

Correctness

Loosely speaking, the protocol is *correct* if the distribution ensembles of its *outputs* in a *real setting* (i.e., in the ASYMMETRIC architecture) are computationally indistinguishable from the distribution ensembles of the enforced algorithm's outputs in an *ideal setting* (i.e., computed by a trusted third party).

Definition 8 (Correct Computation)**.** Let $\mathbb{A}$ be a centralized publishing algorithm, π be the protocol instanciating it over the ASYMMETRIC architecture, and $\mathcal{D}$ be an input dataset. Given $\mathcal{D}$, let $\texttt{REAL} - \texttt{OUT}_{\pi}(\mathcal{D})$ denote the corresponding output distribution in the real setting, and $\texttt{IDEAL} - \texttt{OUT}_{\mathbb{A}}(\mathcal{D})$ denote the corresponding output distribution in the ideal setting. We say that π *correctly computes* $\mathbb{A}$ if:

$$\{\texttt{REAL} - \texttt{OUT}_{\pi}(\mathcal{D})\}_{\mathcal{D}} \stackrel{c}{\equiv} \{\texttt{IDEAL} - \texttt{OUT}_{\mathbb{A}}(\mathcal{D})\}_{\mathcal{D}}$$

Security

The security model must take into account the adversary and his background knowledge. In our context the adversary is the publisher itself, and consequently accesses the transient variables of the execution sequence in addition to the output of the protocol. In the following, we make the standard practical assumption that the adversary is computationally-bounded and model it as a probabilistic polynomial-time algorithm.

In order to enable the participation of the publisher in the protocol, tokens disclose some controlled amount of information to it. The nature of the information voluntarily disclosed depends on the privacy algorithm instanciated, but, since the publisher is untrusted, it is obviously required *by definition* to preserve the privacy guarantees of the algorithm (further details will be given in the following chapters). For example, disclosing the quasi-identifier values both lets the publisher compute k-anonymous equivalence classes through traditional generalization-based algorithms and is considered harmless to the k-ANONYMITY privacy guarantees. In the ideal setting, although the attacker gains knowledge of the sanitized release only, we will assume that he also knows the (harmless) information voluntarily disclosed in the real setting: we are thus able to compare the real and ideal settings based on the *side effect information* disclosed during the execution of the real protocol, excluding the information voluntarily disclosed.

Definition 9 (Secure Computation)**.** Let $\mathbb{A}$ be a centralized publishing algorithm, π be the protocol instanciating it over the ASYMMETRIC architecture, and Δ be the harmless information voluntarily disclosed in the real setting. Given an input dataset $\mathcal{D}$, an attacker $\texttt{A}$ and his arbitrary background knowledge $\chi \in \{0,1\}^*$, we denote $\texttt{REAL} - \texttt{EXEC}_{\pi,\texttt{A}(\chi,\Delta)}(\mathcal{D})$ the distribution representing the adversarial knowledge over the input dataset in the real setting and $\texttt{IDEAL} - \texttt{EXEC}_{\mathbb{A},\texttt{A}(\chi,\Delta)}(\mathcal{D})$ the distribution representing the adversarial knowledge in the ideal setting. We say that π *securely computes* $\mathbb{A}$ if for every adversary $\texttt{A}_r$ attacking π there exists an adversary $\texttt{A}_i$ for the ideal model so that for every $\chi \in \{0,1\}^*$:

$$\{\texttt{REAL} - \texttt{EXEC}_{\pi,\texttt{A}_r(\chi,\Delta)}(\mathcal{D})\}_{\mathcal{D}} \stackrel{c}{\equiv} \{\texttt{IDEAL} - \texttt{EXEC}_{\mathbb{A},\texttt{A}_i(\chi,\Delta)}(\mathcal{D})\}_{\mathcal{D}}$$

Symbol	Meaning
Θ	Population of tokens
n_Θ	Number of tokens in Θ
$\mathcal{D}$	Raw Initial Dataset
$n_\mathcal{D}$	Number of raw records in $\mathcal{D}$
$\mathcal{D}_o$	Obfuscated Collected Dataset
$n_{\mathcal{D}_o}$	Number of tuples in $\mathcal{D}_o$
$\mathcal{R}$	Sanitization Rules
$\mathcal{V}$	Sanitized Release
$n_\mathcal{V}$	Number of sanitized records in $\mathcal{V}$
$\mathtt{E}$, $\mathtt{E}^{-1}$	Encryption, Decryption Schemes
$\mathtt{M}$	MAC Scheme
κ	Cryptographic Material
$\mathtt{H}$	Hash Scheme
$\mathtt{S}$	Sanitization Action

Table 3.2: Transversal Notations

Loosely speaking, the protocol securely computes the centralized publishing algorithm to enforce if the side-effect information disclosed during the protocol's execution does not increase significantly the adversarial knowledge.

3.3.5 Transversal Notations

Table 3.2 summarizes the notations used all along the thesis. The notations specific to each chapter will similarly be summarized within their corresponding chapter.

3.4 Problem Statement

We are now ready to formulate clearly the approach that we suggest in order to achieve the **Decentralization**, **Genericity**, and **Generality** objectives in the context of single-release centralized publishing models and algorithms.

The goal of this thesis is to design a *meta-protocol*, called Meta-Protocol, such that: (1) it is executed on the Asymmetric architecture, (2) its execution sequence is shown to be *correct* and *secure* and follows the **collection**, **construction**, and **sanitization** phases, (3) where the publisher participates in the **collection** phase and is either honest-but-curious or weakly-malicious, and (4) it is scalable to datasets containing on the order of the million of records.

We reached this goal in two incremental steps. We first restricted the **Genericity** objective to generalization-based algorithms for enforcing the k-Anonymity privacy

model. This resulted in the PROTOGEN suite of protocols (one protocol per attack model) described in Chapter 4. We then devised the META-PROTOCOL suite of protocols by abstracting the key concepts of PROTOGEN in order to meet the full **Genericity** objective.

Chapter 4

The Protogen Suite of Protocols

Summary: *This chapter describes the* Protogen *suite of protocols (meaning protocol for generalization). These protocols perform generalization-based algorithms on the* Asymmetric *architecture while coping with the honest-but-curious and weakly-malicious attack models. The primary contribution achieved through* Protogen *is the initiation of the privacy-preserving data publishing algorithms' study in our context: we re-think the traditional phases of generalization-based algorithms in order to shape the execution sequence, and we protect it against passive and active attacks. The content of this chapter includes (but is not limited to) [Allard11a, Allard11b, Allard11c].*

Contents

4.1 Introduction

The k-Anonymity privacy model is an intuitive definition of privacy, and generalization an intuitive way to achieve it. Both are not only widely studied in the computer science litterature (see Section 2.2 but are also considered by governmental agencies [FCSM05] and industrials[22]. Beyond k-Anonymity, generalization is a key enabler for enforcing numerous privacy models [Machanavajjhala06, Machanavajjhala09, Li10b]. Informally speaking, generalization-based algorithms degrade each tuple by replacing precise values, of a selected set of attributes, by ranges. The k-Anonymity model requires each tuple to be indistinguishable from at least $(k-1)$ other tuples based on their quasi-identifying attributes. It is enforced through generalization by defining the set of attributes to be generalized as being the quasi-identifying attributes. The resulting sanitized dataset consists in a set of equivalence classes, each containing at least k tuples sharing the same, generalized, quasi-identifiers. We refer the reader to Section 2.2 for a presentation of the k-Anonymity privacy model and the generalization-based algorithms.

This chapter describes the Protogen suite of protocols (meaning protocol for generalization), designed to enforce generalization-based privacy algorithms for guaranteeing the k-Anonymity privacy model. Through the Protogen suite, we initiated the work conducted in this thesis with a less stringent **Genericity** requirement (limited to generalization-based algorithms). Chapter 5 will show how the key findings of the Protogen have been generalized to achieve the full **Genericity** objective. But we are jumping ahead of our story. Let us start by presenting the three protocols that form the Protogen suite:

- Protogen$_{(hc)}$ tackles the honest-but-curious attack model and settles the shape of the protocols's execution sequences by re-visiting the traditional phases of generalization-based algorithms and defining the information voluntarily disclosed to the publisher to allows its participation in the construction phase;

- Protogen$_{(wm)}$ tackles the weakly-malicious attack model by defining a small set of safety properties in charge of disabling malicious actions - the basis of any weakly-malicious attack - coming from the publisher. The Protogen$_{(wm)}$ protocol results from integrating the safety properties's implementations into the execution sequence of the Protogen$_{(hc)}$ protocol;

[22] For example, Privacy Analytics™(http://www.privacyanalytics.ca/) develops a tool that produces generalized datasets guaranteeing the k-Anonymity privacy model in order to meet the HIPAA privacy rules [HHS02]

- PROTOGEN$^{\dagger}_{(wm)}$ questions the trust assumption put into tokens by considering that a powerful attacker could succeed in breaching into the secure enclosure of tokens.

The chapter is organised as follows. Section 4.2 focuses on the PROTOGEN$_{(hc)}$ protocol. It redesigns the traditional PPDP phases through an execution sequence proven to be correct and secure. Section 4.3 decribes the PROTOGEN$_{(wm)}$ protocol which follows the PROTOGEN$_{(hc)}$'s execution sequence while embedding the safety properties, previously defined and implemented. The trivial alternatives to these protocols are discussed in Section 4.4. Section 4.5 presents our experiments and demonstrates the practicability of the approach. Section 4.6 investigates the situation where tokens can be broken, resulting in the PROTOGEN$^{\dagger}_{(wm)}$ protocol. Finally, Section 4.7 concludes the chapter by discussing the advantages and limitations of the PROTOGEN suite of protocols.

4.2 Honest-but-Curious Publisher

Let us first consider the simplest attack model, namely *honest-but-curious*, where the publisher is assumed to strictly follow the protocol's execution sequence but makes any inference or offline calculation feasible about the association between quasi-identifiers and sensitive values. The PROTOGEN$_{(hc)}$ protocol presented below copes with such an attacker.

4.2.1 Collection, Construction, and Sanitization Phases

The PROTOGEN$_{(hc)}$ protocol, shown in Algorithm 1, allows the publisher to compute the equivalence classes while still guaranteeing k-ANONYMITY. Note that the content of an equivalence class is now made of collected *tuples* instead of raw *records*; we denote it $\mathcal{E}_i.T$, where $\mathcal{E}_i \in \mathcal{E}$ is an equivalence class.

During the **collection phase**, the tokens that connect send to the publisher tuples of the form $(QI, \mathtt{E}_\kappa(SV))$, where $\mathtt{E}$ denotes a symmetric encryption scheme semantically secure against multiple messages (see Section 2.3) parametrized by the cryptographic material κ shared among tokens (stored in their trustworthy computing environment - see Chapter 3). The cryptographic material, similarly to the k-ANONYMITY privacy parameter, is considered to be installed within tokens before the protocol starts (e.g., pre-installed by the manufacturer or resulting from a distributed protocol - see Chapter 6). The set of quasi-identifier values collected precisely forms the information voluntarily disclosed to the publisher, i.e., Δ in the secure computation definition (Definition 9), in order to permit his participation in the construction phase.

Symbol	Meaning
QI	Set of Attributes Forming the Quasi-Identifier
SV	Attribute Forming the Sensitive Value
$\mathcal{E}$	Set of Equivalence Classes
$n_{\mathcal{E}}$	Number of Equivalence Classes in $\mathcal{E}$
$\mathcal{E}_i.\gamma$	Generalization Node of $\mathcal{E}_i \in \mathcal{E}$
$\mathcal{E}_i.T$	Tuples contained in $\mathcal{E}_i \in \mathcal{E}$
$\mathcal{E}_i.T[QI]$	Projection of $\mathcal{E}_i.T$ on QI where $\mathcal{E}_i \in \mathcal{E}$
$\mathcal{E}_i.T[\mathtt{E}_\kappa(SV)]$	Projection of $\mathcal{E}_i.T$ on $\mathtt{E}_\kappa(SV)$ where $\mathcal{E}_i \in \mathcal{E}$
$\mathcal{E}_i.SV$	Decrypted Sensitive Values of $\mathcal{E}_i \in \mathcal{E}$
$\succeq$	Generalization Relationship

Table 4.1: Notations Used in the PROTOGEN Suite of Protocols

When the publisher decides that the collected sample of quasi-identifier values fits his needs (similarly to traditional sampling in a centralized context), he stops the **collection** phase and launches the **construction** phase, during which it computes the set of equivalence classes, denoted $\mathcal{E}$, based on the quasi-identifier values. The **construction** phase is identical to its traditional counterpart in the trusted server context.

Finally, when a token connects during the **sanitization phase**, the publisher chooses a class not yet sanitized, say $\mathcal{E}_i \in \mathcal{E}$, projects the class's tuples on the encrypted sensitive values and sends the resulting multi-set, denoted $\mathcal{E}_i.T[\mathtt{E}_\kappa(SV)]$, to the connected token. The latter returns to the publisher the decrypted sensitive values in a *random order*; the returned sensitive values are denoted $\mathcal{E}_i.SV$. The random order makes the publisher unable to link a decrypted sensitive value to its encrypted version (and consequently to its corresponding quasi-identifier value) based on its position in the returned result.

4.2.2 Correctness and Security

We now show that PROTOGEN$_{(hc)}$ guarantees the k-ANONYMITY privacy model through generalization-based algorithms *correctly* and *securely* (as stated formally in Section 3.3).

Theorem 1. The PROTOGEN$_{(hc)}$ protocol, in charge of computing a generalization-based algorithm, is *totally correct*.

Proof. The publisher forms the dataset to generalize during the **collection** phase. This essentially amounts to sampling the set of tokens similarly to a trusted server context. The collection phase stops when the dataset meets user-defined constraints (e.g., a sufficient cardinality). The **construction** phase then starts and produces the equivalence classes based on the quasi-identifiers of tuples collected. The **construction**

phase is strictly the same as its usual centralized counterpart. Finally, the **sanitization** phase makes use of *any* connected token to decrypt the sensitive values class per class. The **sanitization** phase stops when enough tokens have been connected for decrypting all sensitive values. Since the **collection**, **construction**, and **sanitization** phases are similar to their centralized counterpart, PROTOGEN$_{(hc)}$ *correctly* computes the generalization-based algorithm; and because tokens connect indefinitely, PROTOGEN$_{(hc)}$ terminates. □

Theorem 2. The PROTOGEN$_{(hc)}$ protocol *securely* computes generalization-based algorithms.

Proof. During the **collection** phase, the only tuples that the publisher has at his disposal are in the form $(QI, \mathtt{E}_\kappa(SV))$. The publisher learns the voluntarily disclosed information, i.e., the set of quasi-identifiers, but has no feasible way to decrypt the sensitive values. The **construction** phase does not bring to the publisher any additional information. Finally, the **sanitization** phase unveils the mapping between the set of tuples of each equivalence class and its corresponding set of returned sensitive values: $\forall \mathcal{E}_i \in \mathcal{E}$ the publisher maps $\mathcal{E}_i.T[QI]$ to $\mathcal{E}_i.SV$. Because (1) the equivalence classes have been formed by strictly following a centralized generalization algorithm (remind that the publisher is honest-but-curious), and (2) the order of sensitive values in the returned set is randomized by tokens, then the mapping between quasi-identifier values and sensitive values in the real setting is strictly the same as the mapping in the ideal setting. □

4.2.3 Execution Sequence

Algorithm 1 details the complete execution sequence of the PROTOGEN$_{(hc)}$ protocol.

4.3 Weakly-Malicious Publisher

A weakly-malicious publisher still aims at inferring data but now deviates from the protocol if he will not be detected and he will be able to produce a correct result. As stated in Chapter 3, he may tamper the collected dataset through the forge, copy, and delete tampering actions or produce unsafe equivalence classes. We disable any tampering of the PROTOGEN's execution sequence by requiring each phase to be made of a single step, thereby precluding by design any attack on the execution sequence integrity.

This section starts by representative examples of attacks. Second, it defines the set of safety properties in charge of preventing any basic tampering action and of

Algorithm 1: PROTOGEN$_{(hc)}$

req. (tokens) : The cryptographic material common to tokens (κ).
req. (publisher): The k-ANONYMITY and generalization parameters (the value of k and the generalization hierarchies), the cardinality of the obfuscated dataset to form ($n_{\mathcal{D}_o}$).
output : The sanitized release $\mathcal{V}$ obtained by generalizing the dataset $\mathcal{D}$ collected during the protocol.

begin Collection phase
  forall $i \in [1, n_{\mathcal{D}_o}]$ **do** Each token that connects (and that did not connect previously during the same **collection** phase) sends a tuple t_i of the form $(QI, \mathtt{E}_\kappa(SV))$ to the publisher.
end
begin Construction phase
  The publisher computes $\mathcal{E}$, the set of equivalence classes. The publisher is honest (though curious), so all classes respect the k-ANONYMITY privacy model for the given k.
end
begin Sanitization phase
  foreach $\mathcal{E}_i \in \mathcal{E}$ **do**
    The publisher sends $\mathcal{E}_i.T[\mathtt{E}_\kappa(SV)]$ to a connecting token $\theta \in \Theta$;
    repeat on θ's side
      Choose randomly an encrypted sensitive value $sv_e \in \mathcal{E}_i.T[\mathtt{E}_\kappa(SV)]$;
      Delete it from $\mathcal{E}_i.T[\mathtt{E}_\kappa(SV)]$ (locally);
      Decrypt sv_e: $sv \leftarrow \mathtt{E}_\kappa^{-1}(sv_e)$;
      Return sv to the publisher;
    until $\mathcal{E}_i.T[\mathtt{E}_\kappa(SV)] = \varnothing$;
end
return The sanitized release $\mathcal{V}$ made of the set of equivalence classes $\mathcal{E}$ where the set of tuples of each class has been replaced by the class's decrypted sensitive values.

checking the safety of classes; enforcing safety properties is then shown to enable a correct and secure protocol (provided that their implementations preserves correctness and security). Fourth, implementations of safety properties are proposed. Finally, we describe the complete sanitization phase of the PROTOGEN$_{(wm)}$ protocol, which is the result of integrating the safety properties's implementations into the PROTOGEN$_{(hc)}$ protocol .

4.3.1 Examples of Weakly-Malicious Attacks

We illustrate now the weakly-malicious attacks which could be conducted by the publisher through simple examples that cover all weakly-malicious actions.

Unsafe Equivalence Classes and Forge Based Attacks

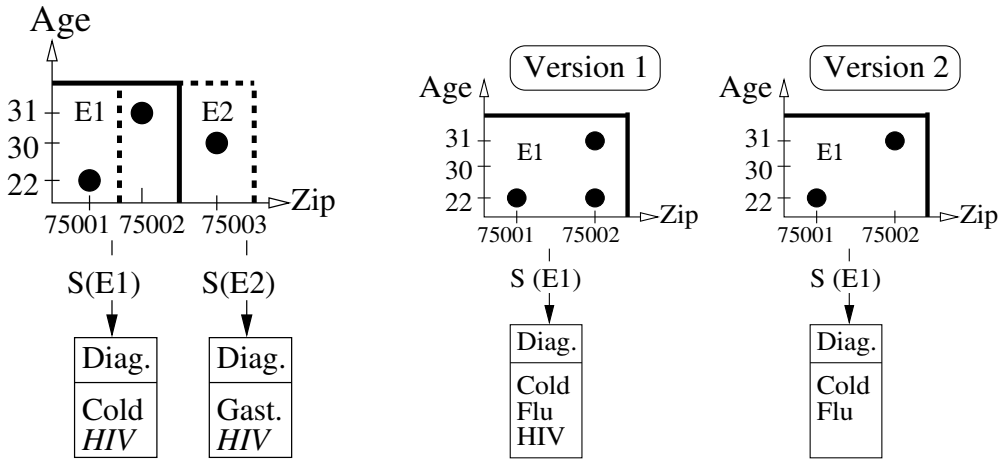
A simplistic attack from the publisher consists in constructing equivalence classes that contain less than k tuples: they are simply not k-anonymous. This attack can be further refined by forging tuples and inserting them into classes such that they contain more than k tuples, but actually less than k *collected* tuples.

Copy/Delete Based Attacks

Tuples can be copied in two ways: either the publisher produces a class that contains copies of the same set of tuples (called *intra-class copy*), or he produces two classes, one containing a subset of tuples from the other (called *inter-class copy*). Intra-class copies lead to a direct reduction of the k-ANONYMITY level of the class, as previous actions do. Inter-class copies lead to inferences that are based on computing the differences between their respective sensitive values and quasi-identifier values. Indeed, (1) the sensitive values returned for both classes correspond to the quasi-identifier values belonging to both classes (the copied subset of tuples), and (2) the sensitive values returned for only one class correspond to the quasi-identifier values belonging to that class only. After having computed the differences, the publisher is thus able to draw a correspondence between subsets of quasi-identifier values and subsets of sensitive values. These attacks are called *differential attacks*.

Figure 4.1(a) depicts a differential attack based on an inter-class copy tampering action. In this example, the two equivalence classes $\mathcal{E}_1$ (the class depicted in plain lines) and $\mathcal{E}_2$ (the class depicted in dashed lines) contain a tuple in common, resulting from a copy between $\mathcal{E}_1$ and $\mathcal{E}_2$. Since the publisher built the classes, he knows the only quasi-identifier value that appears in both classes, which corresponds to the only sensitive value that appears in both returned sets: $(QI = (75002, 31), SV = HIV)$.

He also knows the only quasi-identifier value that appears in $\mathcal{E}_1$ and which corresponds to the sensitive value that appears only in $\mathcal{E}_1$'s decrypted sensitive values: $(QI = (75001, 22), SV = Cold)$; similarly with $\mathcal{E}_2$, he learns that $(QI = (75003, 30), SV = Gast.)$.



(a) Based on an Inter-Copy Action

(b) Based on a Delete Action

Figure 4.1: Examples of Differential Attacks

As illustrated in Figure 4.1(b), delete actions also lead to differential attacks: the publisher produces two versions of a class, deletes a tuple in the second version, and sanitizes both versions. The sensitive value corresponding to the deleted quasi-identifier value will be the one that appears only in the sanitized result of the first version. Note the similarity between delete actions and inter-class copy actions: both yield the mapping between *subsets* of quasi-identifier values and sensitive values.

4.3.2 Safety Properties

Having motivated the need to protect from weakly-malicious actions, we investigate the *safety properties* required to preclude all of them. Safety properties complement each other: each of them precludes a precise weakly-malicious action, and all of them enforced together preclude any combination of weakly-malicious actions. By precluding basic actions (which are only a few), safety properties discard any possibility of attacks (which consist of arbitrary combinations of basic actions).

Preclude Unsafe Classes

The `Safe Rules` safety property (Def. 10) is in charge of precluding the sanitization of classes that contain less than k tuples.

Definition 10 (`Safe Rules`)**.** Let $\mathcal{E}_i \in \mathcal{E}$, then $|\mathcal{E}_i.T| \geq k$.

Preclude Forge Actions

The *forge* tampering action allows the attacker to forge and inject tuples into the dataset (such that, e.g., an equivalence class - though unsafe - contains more than k tuples). The `Origin` safety property states that a collected tuple must be accompanied with its signature as a proof of origin and integrity. We can easily observe in Definition 11 that only tokens can produce the signature because the attacker has no access to the cryptographic material.

Definition 11 (`Origin`)**.** Let $(b, \sigma) \in \{0,1\}^* \times \{0,1\}^n$ be a bitstring (i.e., any data structure) with its n-bit secret key signature. The data structure b respects the origin property if $\sigma = \mathtt{M}_\kappa(b)$, where $\mathtt{M}_\kappa$ is the message authentication code scheme parametrized by the token's cryptographic material.

Consequently, each collected tuple t embeds its (secret key) signature, denoted $t.\sigma$. The detailed input of the MAC scheme is given in the implementation section below.

Preclude Copy Actions

Let focus first on intra-copy actions by assuming that the construction phase results in a single equivalence class. The `Identifier Unicity` safety property (Def. 12) precludes intra-copy actions by requiring each tuple to be assigned a unique identifier and guaranteeing that each tuple identifier of the class be unique.

Definition 12 (`Identifier Unicity`)**.** Let $t \in \mathcal{E}_i.T$ be a tuple in the equivalence class $\mathcal{E}_i \in \mathcal{E}$, and $t.\tau$ denote t's identifier. The class $\mathcal{E}_i$ respects the `Identifier Unicity` safety property if for every pair of tuples $t_m, t_n \in \mathcal{E}_i.T$, $m \neq n \Rightarrow t_m.\tau \neq t_n.\tau$.

Now, let consider several equivalence classes. In addition to being unique in its own class, each tuple must also appear in a single class. First, the `Membership` safety property (Def. 13) states that each tuple's quasi-identifier value specializes the generalization node of the tuple's equivalence class. Second, the `Mutual Exclusion` safety property (Def. 14) ensures that each tuple's quasi-identifier value specializes (the generalization node of) a single class. We observe that `Mutual Exclusion` precludes constructing the equivalence classes based on a local recoding algorithm because it may

generate classes whose generalization nodes overlap (see Section 2.2 for a description of generalization-based algorithms).

Definition 13 (`Membership`)**.** An equivalence class $\mathcal{E}_i \in \mathcal{E}$ respects `Membership` if for every tuple $t_j \in \mathcal{E}_i.T$, then $t_j.QI \preceq \mathcal{E}_i.\gamma$.

Definition 14 (`Mutual Exclusion`)**.** A set of equivalence classes $\mathcal{E}$ respects `Mutual Exclusion` if for every pair of classes $\mathcal{E}_i, \mathcal{E}_j \in \mathcal{E}$, $i \neq j \Rightarrow \mathcal{E}_i.\gamma \cap \mathcal{E}_j.\gamma = \varnothing$.

We can easily observe that the `Identifier Unicity`, `Membership`, and `Mutual Exclusion` properties together guarantee the absence of intra/inter-class Copy actions.

Theorem 3. Enforcing together the `Identifier Unicity`, `Membership`, and `Mutual Exclusion` safety properties is necessary and sufficient for guaranteeing the absence of Copy action.

Proof. We start by showing the sufficency of these properties. First, `Identifier Unicity` is sufficient to preclude by itself intra-class copy actions (recall that the integrity of a tuple identifier is guaranteed by the safety property in charge of precluding forge-based attacks). Second, assume that a given tuple t has been copied into two distinct classes. Only one of them generalizes t's quasi-identifier value because otherwise `Mutual Exclusion` would be contradicted. Consequently there must be one class that does not generalize t's quasi-identifier value. This contradicts the `Membership` safety property. As a result, `Membership` and `Mutual Exclusion` are together sufficient for precluding inter-class copy actions.

We now show the necessity of these properties. First, since a distinct identifier is assigned to each tuple, the absence of intra-class copy results immediately in the satisfaction of the `Identifier Unicity` property. Second, the absence of inter-class copy, implies that: (1) the `Mutual Exclusion` property is satisfied in that the classes's generalization nodes do not overlap (a property of global recoding generalization algorithms) and (2) the `Membership` property too in that each tuple appears in the class which generalization node generalizes its identifier. □

Preclude Delete Actions

An equivalence class is affected by a *delete* tampering action if its generalization node has been associated to (at least two) distinct sets of tuples. To avoid such actions, the content of equivalence classes must not change along the sanitization phase: it must be made *invariant*. We define below the `Invariance` property such that it can adapt to various data structures.

Definition 15 (`Invariance`)**.** Let $LABEL$ be a set of character strings containing the labels designing the data structures to be made invariant; $LABEL$ is known by both the publisher and the tokens. Let $l_0 \in LABEL$ 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 token 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 token, then $\texttt{Pr}[(l_0, b_i)] = 1 - \epsilon$ if $b_i = b_0$, and $\texttt{Pr}[(l_0, b_i)] = \epsilon$ otherwise, where $\epsilon \in]0, 1]$ is a customizable parameter.

For example, the set of equivalence classes is invariant if the couple ("$\mathcal{E}$", $\mathcal{E}$) respects the `Invariance` safety property, "$\mathcal{E}$" being the label and $\mathcal{E}$ the actual bitstring representation. The implementation section will detail how the `Invariance` of the set of equivalence classes is actually enforced.

Why is the `Invariance` safety property probabilistic rather than certain? Recall that the phases of the PROTOGEN$_{(wm)}$ protocol are required to be performed within a single execution step. Defining `Invariance` as probabilistic gives us additional flexibility for making its implementation meet the single execution step constraint.

4.3.3 Correctness and Security Analysis

We now analyze the correctness and security guarantees given by the PROTOGEN$_{(wm)}$ protocol.

Correctness

The PROTOGEN$_{(wm)}$ protocol is merely the PROTOGEN$_{(hc)}$ protocol into which the implementations of safety properties have been integrated. Since the safety properties do not alter the execution sequence, PROTOGEN$_{(wm)}$ is also totally correct.

Theorem 4. The PROTOGEN$_{(wm)}$ protocol, in charge of computing the generalization algorithm, is *totally correct*.

Proof. The dataset formed during the collection phase still contains the quasi-identifier values in clear and the sensitive values encrypted. The PROTOGEN$_{(wm)}$'s collection and construction phases differ from the PROTOGEN$_{(hc)}$'s only in that tuples embed additional information (i.e., identifiers and signatures). These phases are thus correct. During the sanitization phase, the implementations of safety properties are executed. Nevertheless, since they are required *by definition* to preserve the protocol's correctness, the sanitization phase is correct. All phases are correct and terminate, so PROTOGEN$_{(wm)}$ is totally correct. □

Probabilistic Secure Computation

The `Invariance` property is voluntarily probabilistic. This impacts the secure computation definition; we relax it for PROTOGEN$_{(wm)}$ by making it probabilistic too.

Definition 16 (Probabilistic Secure Computation)**.** Let $\mathbb{A}$ be a centralized publishing algorithm and π be the protocol instanciating it over the ASYMMETRIC architecture. We say that π *probabilistically securely computes* $\mathbb{A}$ if π securely computes $\mathbb{A}$ **with probability** $(1-\epsilon)$ **- where** $\epsilon \in]0,1]$ **is a customizable parameter close to** 0.

Theorem 5. The PROTOGEN$_{(wm)}$ protocol, in charge of computing a generalization-based algorithm, is *probabilistically secure.*

Proof. A computationally-bounded weakly-malicious attacker cannot launch any active attack because he would have to use weakly-malicious actions as building blocks. As such actions will be detected by tokens (probabilistically for the delete action, and with certainty for the others), and because he is reluctant to being convicted as an adversary, he is reduced to launching passive attacks only. Since the security of the execution sequence against passive attacks has already been shown in Section 4.2, and because safety properties's implementations are required *by definition* to preserve the protocol's security, then PROTOGEN$_{(wm)}$ is probabilistically secure. □

4.3.4 Safety Properties Implementation

We observe that safety properties divide in two groups. *Local properties* concern the content of each equivalence class, independently from the others - they consist of the `Safe Rules`, `Identifier Unicity`, `Origin`, and `Membership` properties - and *global properties* concern the whole set of classes sent to tokens - they consist of the `Mutual Exclusion` and `Invariance` properties.

Local Properties

Checking the local properties in a token is rather straightforward. To test the `Safe Rules` property, each token receiving an equivalence class during the Sanitization phase checks that the number of tuples in the class is greater than k. The `Identifier Unicity` is easily fulfilled by defining the tuple identifier as being, e.g., the semantically secure encryption of an identifier proper to each token, and checking that within a received class each identifier is unique. Identifiers cannot be tampered by the publisher: being part of tuples their integrity is guaranteed by the `Origin` safety property. To check the `Membership` property, tokens participating in the sanitization phase download the class's generalization node and the tuples' quasi-identifier values, in addition

to the set of sensitive values (see below how the `Mutual Exclusion` implementation completes the enforcement of `Membership`). They can now check whether all the quasi-identifier values received specialize their class's generalization node. Finally, the `Origin` property is asserted by signing each collected tuple completely (e.g., signing the bitwise concatenation of the tuple's quasi-identifier value, sensitive value, and identifier).

The implementations of local properties only consist of checks: they have no impact on the execution sequence. They consequently preserve the protocol's correctness. The additional information disclosed to the publisher consists of the tuples's identifiers and the tuples's signatures, which are indistinguishable from random numbers, and therefore also preserve the protocol's security.

Global Properties

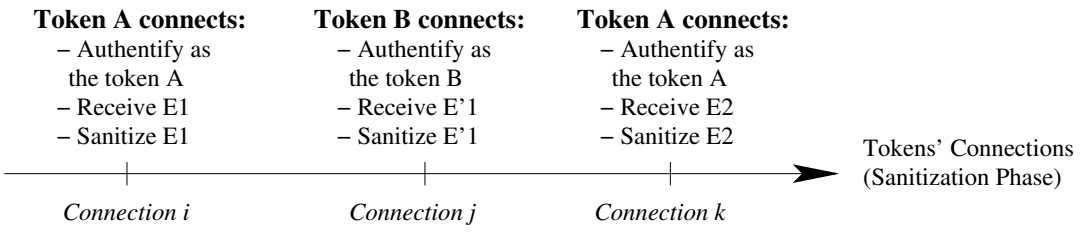


Figure 4.2: No Caveat Action

Each token receives a single equivalence class per session, so checking the global properties *would* require that tokens share information among them about the classes received. Unfortunately, tokens are not able to communicate together, neither directly with each other (they are disconnected devices), nor indirectly through the publisher (this would require more than one execution step); each token has to rely on its own history only. As a result, the publisher can easily select the equivalence class sent to each token such that all the properties are satisfied from the token's viewpoint while they are violated from a global viewpoint. Figure 4.2 depicts this situation. The publisher sends to token A the class $\mathcal{E}_1$, and to token B the class $\mathcal{E}'_1$ which is the result of tampering $\mathcal{E}_1$ by a delete action. The detection probability is null because the publisher will not send $\mathcal{E}'_1$ to the token able to detect the deletion, i.e., token A. Similarly, the classes $\mathcal{E}_1$ and $\mathcal{E}_2$ may be affected by an inter-copy action without tokens A and B able to detect it. Considering that the publisher has weakly-malicious intents, we propose to deter him from violating global properties by making any violation visible to tokens through *caveat actions*.

Caveat Actions. The first caveat action is to use anonymous communication channels [Ren10] (e.g., mixnets) between the publisher and tokens. This precludes the

publisher to control which token receives which equivalence class. Consequently, the probability to send classes violating the global properties to the same token is no longer null. As depicted in Figure 4.3, the publisher does not know whether the connections i and j are due to the same token or not: the detection probability is not null.

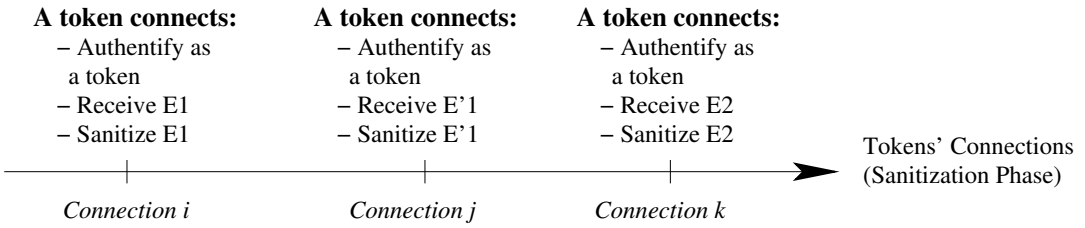


Figure 4.3: After the First Caveat Action

To increase the detection probability, the second caveat action allows the attack detection to occur during the complete sanitization phase, not anymore during the sanitization of the contradicting class only. For the presentation's clarity, let us start by considering `Invariance` only. The second caveat action forces the publisher to produce a *summary* of the equivalence classes (denoted $\mathcal{S}$ hereafter) which contains for every class a hash of its tuples (also called a *digest*), and to send it to each token that connects. Each participating token thus acquires a global view over the equivalence classes through the summary. Now, detecting an attack does not require that the same token sanitizes both versions of the tampered class: any token that has received two summaries disagreeing on the content of a class detects the attack. Figure 4.4 depicts the situation. The class $\mathcal{E}_1$ is under attack. In this example, the sanitization phase is split into three times as follows (in the general case, they do not occur necessarily in this order): (1) the time t_1 during which $\mathcal{E}_1$ is sanitized (in Fig. 4.4, t_1 consists in the connection i), (2) the time t_2 during which $\mathcal{E}'_1$ is sanitized (in Fig. 4.4, t_2 consists in the connection j), and (3) the time t_r that encompasses the rest of the sanitization phase (in Fig. 4.4, the connection k is part of t_r as any connection other than i and j). We easily observe that: (1) during t_1, the summary contains the digest of $\mathcal{E}_1$ (the publisher has no alternative because the token sanitizing $\mathcal{E}_1$ checks that the summary contains $\mathcal{E}_1$'s actual digest), (2) during t_2, the summary contains the digest of $\mathcal{E}'_1$ (similarly, the publisher has no alternative), and (3) during t_r, the summary contains one of the two digests. The attack is detected if the same token receives two summaries that contain different digest(s). In the example, the token having received the summary containing $\mathcal{E}'_1$'s digest (during t_2) detects the attack if it connects either during t_1 or during t_r.

Let us now consider `Mutual Exclusion` again. By adding the generalization nodes of classes into the summary, tokens can also check that classes are mutually exclusive (we discuss the algorithm below). Although the second caveat action increases the

detection probability, it is still likely to be insufficient in practice.

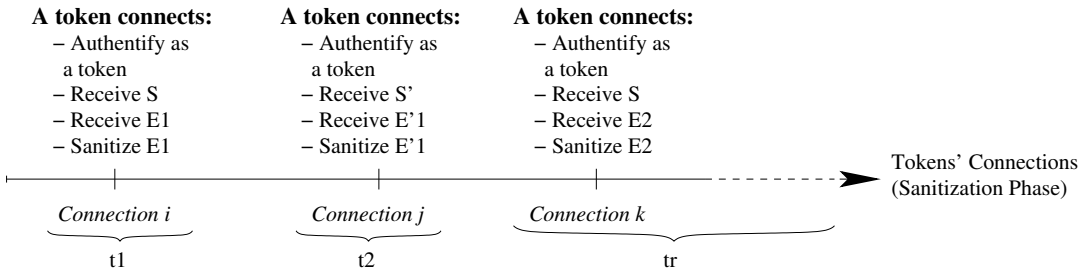


Figure 4.4: After the Second Caveat Action

So the third and last caveat action is to force the publisher to send each class and corresponding summary to a predefined number of tokens: the detection probability does not rely anymore on the probability that a single token connects during t_r but depends on the number of tokens receiving each class. The detection probability is now customizable. Figure 4.5 illustrates the third caveat action, where four tokens are required to receive each class.

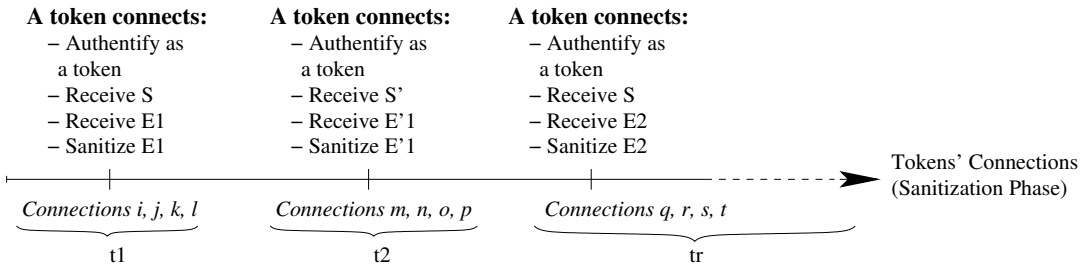


Figure 4.5: After the Third Caveat Action

Detection Probability. Let n_{min} denote the minimum number of tokens receiving each class and corresponding summary; we (realistically) assume that this number is much lower than the total number of tokens: $n_{min} \ll n_\Theta$. The most favorable case for the attack is when the publisher produces a minimal number of disagreeing summaries, i.e., two versions. Let assume that the summary used during the rest of the sanitization phase is t_1's summary. Such an attack is not detected if the tokens receiving t_2's summary connect only during t_2. Consequently, the detection probability depends on the probability of randomly choosing a token that connects only during t_2. We denote this probability $\texttt{Pr}_{t_2}$. For the moment, let assume that we know $\texttt{Pr}_{t_2}$; we explain below how to compute it.

Our goal here is to determine n_{min} based on the minimal detection probability to guarantee (a parameter of the protocol). The minimal detection probability, denoted

$\texttt{Pr}_{min}$(detect), is the opposite probability of randomly choosing n_{min} tokens that connect only during t_2: $\texttt{Pr}_{min}(\text{detect}) = 1 - (\texttt{Pr}_{t_2})^{n_{min}}$. Consequently

$$n_{min} = \frac{ln(1 - \texttt{Pr}_{\min}(\text{detect}))}{ln(\texttt{Pr}_{t_2})} \tag{4.1}$$

For instance, even for highly unfavorable (and unrealistic) values such $\texttt{Pr}_{t_2} = 0.8$, $n_{min} = 21$ guarantees a minimal probability of detection $\texttt{Pr}_{min}(detect) \approx 0.991$.

How is $\texttt{Pr}_{t_2}$ computed? The model underlying the detection probability depends on the connection behavior of tokens, which depends on the actual application. $\texttt{Pr}_{t_2}$ is thus computed by first modeling the distribution representing the connection probability of tokens (based on external knowledge such as, e.g., the profiles of tokens' holders). For simplicity, we assume this distribution to be independent of time (i.e., whatever the connection time the tokens's connection probabilities are the same); extending the model to time-dependent distributions is straightforward. We denote D the distribution probability of tokens' connections, $D(\theta)$ denoting the connection probability of token θ. Second, the probability of each token to connect only during t_2 is computed based on the distribution D, the number of tokens connections required to perform the complete sanitization (denoted l), and the number of tokens connections during t_2 (which is n_{min}). Let denote $\texttt{Pr}(t_2|D(\theta), l, n_{min})$ the probability that token θ connects only during t_2. The probability of randomly choosing a token that connects only during t_2 is thus: $\texttt{Pr}_{t_2} = \sum_{\forall\theta} D(\theta) \cdot \texttt{Pr}(t_2|D(\theta), l, n_{min})$ so

$$\texttt{Pr}_{t_2} = \sum_{\forall\theta} D(\theta) \cdot (1 - D(\theta))^{l - n_{min}} \tag{4.2}$$

Similarly to the cryptographic material and privacy parameters, we consider that n_{min} is computed beforehand and pre-stored within the secure environment of tokens before the protocol starts. When receiving a class to sanitize, the given token only sanitizes and returns the $(n_{min})^{th}$ fraction of the tuples received so that n_{min} other tokens are necessary to complete the class's sanitization. To avoid sanitizing several times the same tuples, a straightforward duplicate handling scheme can be implemented based on the MACs of the tuple identifiers. It is important to note that n_{min} can be arbitrarily big, and in particular, be greater than the cardinality of equivalence classes: each class and corresponding summary are sent to n_{min} tokens, that enforce the safety properties but do not necessarily return any tuple. High probabilistic detection comes at the cost of sending the equivalence classes to more tokens.

Correctness And Security Preserved. Similarly to local properties's implementations the `Mutual Exclusion`'s implementation is only a check; it has no impact

on the execution sequence so obviously preserves its correctness. With the `Invariance`'s implementation however, classes are sanitized partially by each connecting token. Nevertheless, a trivial duplicate detection mechanism allows tokens to identify in each class the tuples that have already been sanitized without disclosing any significant information to the publisher. The global properties's implementations consequently preserve the correctness and security of the protocol.

Checking Mutual Exclusion Efficiently. Although smart implementations of the `Mutual Exclusion` property can be designed in order to avoid a nested-loop style comparison of classes (e.g., in a sort-merge fashion), `Mutual Exclusion` remains one of the most costly checks. The cost is due to the necessity of checking the absence of overlap across the various dimensions of the quasi-identifier. However, by slightly extending the `Invariance` property to encompass the classes's nodes in addition to their content, we can avoid the cost of checking `Mutual Exclusion` between summaries received at different moments. Indeed, if during its first connection, a token checks that the summary asserts `Mutual Exclusion`, it has only to check that the summary never changes during its following connections to guarantee that classes never overlap.

4.3.5 The Protogen$_{(wm)}$ Execution Steps

Algorithm 2 details the sanitization phase of the algorithm to be executed by each token. If a property check is not fulfilled, the token stops the execution and raises an alarm (e.g., to the destination of the token owner or a trusted third party).

4.4 Inadequacy of Trivial Solutions

Trivial alternatives to the PROTOGEN$_{(hc)}$ and PROTOGEN$_{(wm)}$ protocols could be devised based on collecting quasi-identifiers and sensitive values separately. Indeed, one could imagine to reorganize the phases as follows: (1) a phase of *quasi-identifier collection*, during which tuples collected only consist of quasi-identifiers, followed by (2) a phase of *construction* similar to the PROTOGEN's construction phase, and (3) a phase of *sensitive values collection* during which each token that connects downloads the set of generalization nodes of all classes and returns its sensitive value and the generalization node that generalizes its quasi-identifier value.

We denote the above scheme as *protonaive*. Although protonaive tackles the honest-but-curious attack model without requiring the sharing of any cryptographic material between tokens, it still needs safety properties to cope with a weakly-malicious publisher. Moreover, protonaive incurs either an unbounded latency (if the publisher

Algorithm 2: Sanitization Phase of PROTOGEN$_{(wm)}$ - Token Side

requirements: An anonymous communication channel between the tokens and the publisher, the k-ANONYMITY and generalization parameters (k, the generalization hierarchies), the cryptographic material common to tokens (κ), the minimal number of tokens that must receive each class n_{min}.

output : A set of decrypted sensitive values or nothing.

Receive the current Summary $\mathcal{S}$, where $\mathcal{S}.\Delta$ denote the classes' digests and $\mathcal{S}.\Gamma$ the classes' generalization nodes, and $\delta_i \in \mathcal{S}.\Delta$ and $\gamma_i \in \mathcal{S}.\Gamma$ respectively denote the digest and generalization node of the class $\mathcal{E}_i \in \mathcal{E}$;

if $\nexists$ *previous summary* $\mathcal{S}_p$ **then**

  Check the `Mutual Exclusion` property: $\forall \delta_i, \delta_j \in \mathcal{S}.\Delta,\ \delta_i \neq \delta_j \Rightarrow \delta_i \cap \delta_j = \varnothing$;

else

  Check the `Invariance` of the number of classes: $|\mathcal{S}.\Delta| = |\mathcal{S}.\Gamma| = |\mathcal{S}_p.\Delta| = |\mathcal{S}_p.\Gamma|$;

  Check the `Invariance` of the classes's digests: $\forall \delta_i \in \mathcal{S}.\Delta \exists \delta_j \in \mathcal{S}_p.\Delta$ s.t. $\delta_j = \delta_i$;

  Check the `Invariance` of the classes's nodes: $\forall \gamma_i \in \mathcal{S}.\Gamma \exists \gamma_j \in \mathcal{S}_p.\Gamma$ s.t. $\gamma_j = \gamma_i$;

Download a class $\mathcal{E}_i \in \mathcal{E}$ and the MACs of the class's tuples already sanitized $\mathcal{E}_i.M$;

Check the consistency between the summary and the class: $\delta_i = \mathtt{H}(\mathcal{E}_i.T)$ and $\gamma_i = \mathcal{E}_i.\gamma$;

Check the `Safe Rules` property: $|\mathcal{E}_i.T| \geq k$;

Init. the TIDs sets: $I \leftarrow \varnothing$;

foreach $t \leftarrow (QI, \mathtt{E}_\kappa(SV), \tau, \sigma) \in \mathcal{E}_i.T$ **do**

  Check the `Origin` property: $\mathtt{M}_\kappa(QI||SV||\tau) = t.\sigma$;

  Check the `Membership` property: $t.QI \preceq \mathcal{E}_i.\gamma$;

  Check the `Identifier Unicity` property: $t.\tau \notin I$;

  $I \leftarrow I \cup t.\tau$;

if $|\mathcal{E}_i.T|/n_{min} \geq 1$ **then**

  return *$|\mathcal{E}_i.T|/n_{min}$ tuples randomly chosen in $\mathcal{E}_i.T$ with the MAC of their identifier, s.t. the MAC does not appear in $\mathcal{E}_i.M$.*

else

  return *With probability $|\mathcal{E}_i.T|/n_{min}$, return one random tuple in $\mathcal{E}_i.T$ with the MAC of its identifier, s.t. the MAC does not appear in $\mathcal{E}_i.M$.*

wishes to collect the sensitive values of *all* the tokens having participated to the quasi-identifier collection), or a discrepancy between the collected quasi-identifiers and sensitive values (otherwise). Avoiding these drawbacks motivates the proposed PROTOGEN protocols, that collect both quasi-identifiers and sensitive values during the first phase.

4.5 Experimental Validation

4.5.1 Experimental platform

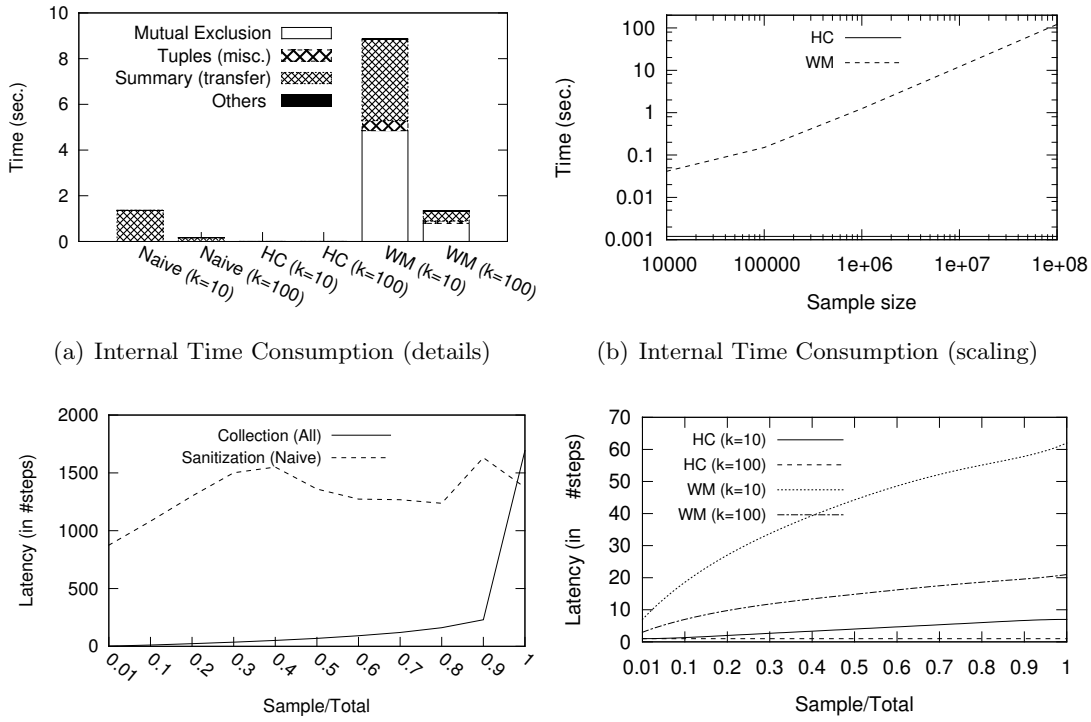
The protocols presented in this paper have been implemented and are being integrated in the PlugDB prototype [Anciaux10] described in Chapter 6. The hardware platform is provided by Gemalto (the world leader in smart-cards), industrial partner of the project. The hardware platform is still under test so the performance measurements have been conducted on a cycle-accurate hardware emulator. The protocols considered for the experiments are PROTOGEN$_{(hc)}$ and PROTOGEN$_{(wm)}$. As a comparison baseline we also implemented protonaive, the trivial protocol described in Section 4.4. These protocols are denoted respectively HC, WM, and Naive in the figures.

We concentrate on the evaluation (1) of the time spent internally in each token to participate to each phase of the protocol, (2) of the protocol latency, and (3) of the detection probability of an attack on the global properties. We obtained the results of point (1) by performance measurements conducted on the hardware emulator, and the results of points (2) and (3) by simulation.

4.5.2 Internal Time Consumption

Settings

Let us briefly summarize the main settings concerning the token. The cycle-accurate hardware simulator we used for this experiment is clocked at 50Mhz, corresponding to the CPU clock of the target platform. Cryptographic operations are implemented in hardware with good performances (e.g., encrypting a block of 128bits with AES costs 150 cycles). Although Hi-Speed USB2 (480 Mbps theoretical bandwidth) is announced for the near future, today's implementation of the communication channel is far less efficient. The measured throughput is 12Mbps (i.e., Full-Speed USB2), which amounts to 8Mbps of useful bandwidth when we exclude the overhead of the USB protocol itself.



(a) Internal Time Consumption (details)

(b) Internal Time Consumption (scaling)

(c) Latencies: Collection Phase (all) and Sanitization Phase (protonaive)

(d) Latencies: Sanitization Phases of PROTOGEN$_{(hc)}$ and PROTOGEN$_{(wm)}$

Figure 4.6: token's Time Consumption

Internal time consumption

Figure 4.6(a) details the time consumed by a token for each basic operation performed during the sanitization phase. The measure has been performed with a sample of 10^6 tokens, k varying from 10 to 100. The dataset was synthetically generated; two numerical attributes formed the quasi-identifier and one string attribute formed the sensitive value.

Depending on the protocol, the worst case occurs either when k is minimal or maximal. For each protocol, we plot these two cases to assess whether performance bottlenecks could compromise the feasibility of the approach. The worst case for protonaive and PROTOGEN$_{(wm)}$ occurs when k is low. In this situation, the transfer cost of the summary (for both) and the cost of checking `Mutual Exclusion` (for PROTOGEN$_{(wm)}$) dominate the other costs because of the high number of equivalence classes. Note that a token only checks `Mutual Exclusion` once, i.e., at its first connection. It then checks the `Invariance` of generalization nodes during its subsequent connections. Operations

related to tuples (i.e., transfer, hashing, and decryption) are cheap since protonaive solely uploads its sensitive data, and PROTOGEN$_{(wm)}$ downloads and uploads between k and $2k-1$ tuples. On the contrary, the worst case for PROTOGEN$_{(hc)}$ occurs for a high k value, where the tuples' transfer cost overwhelms the other costs. Indeed, since PROTOGEN$_{(hc)}$ does not make use of any summary, its cost is not impacted by the number of classes but only by the cardinality of each class. As a conclusion, this figure confirms the feasibility of the approach by showing that, even in the worst cases, the execution time amounts to couples of seconds.

Scaling

Figure 4.6(b) shows the scaling of all the protocols with respect to the number of tokens in the sample - chosen to be on a nation-wide scale - with $k = 100$. Apparently, protonaive and PROTOGEN$_{(wm)}$ scale linearly with the number of tokens sampled. This is due to the linear increase in the number of classes (cost of transferring the summary and checking the global properties). The cost of the PROTOGEN$_{(hc)}$ protocol remains constant, around 10^{-3} sec.; indeed, it does not use any summary so the time consumed by the sanitization of a class only depends on the class's cardinality and the tuple's size.

4.5.3 Latencies

Figures 4.6(c) and 4.6(d) plot respectively the latency of the collection and sanitization phases of the protocols, considering a population of $n_\Theta = 1$ million tokens. The latency is measured in terms of connection steps of equal duration, this duration being application dependent. We consider that the connection probability of tokens, i.e., D, is uniform, and that 1% of the tokens population connects at each step.

Collection Phase (All) / Sanitization Phase (Naive)

The latency of the collection phase is the same, regardless of the protocol studied. This latency depends on the distribution of connection probabilities and on the proportion of tokens in the sample. Figure 4.6(c) shows that the latency is about 160 steps when considering a sample of 80% of the tokens population. Note that the latency depends primarily on the connection probabilities of tokens (and not on the total number of tokens): the less often the tokens connect, the longer the collection phase will be. On the same figure, since the times involved are of the same magnitude, we have also plotted the latency of the sanitization phase of the protonaive protocol. This latency is about 1000 steps, regardless of the proportion of tokens reconnecting (and would

be even bigger for a skewed distribution). These high numbers are explained by the fact that the *same* tokens must connect during both the quasi-identifier values and the sensitive values collection phases.

Sanitization Phase

Figure 4.6(d) shows the latency of the sanitization phase of the PROTOGEN$_{(hc)}$ and PROTOGEN$_{(wm)}$ protocols for $k = 10$ and $k = 100$. For PROTOGEN$_{(hc)}$, we assumed that a connecting token sanitizes exactly one class during its session. The latency is linear and depends on the total number of classes to sanitize divided by the number of connected tokens per step. The PROTOGEN$_{(hc)}$'s latency is constant and equal to 1 for $k = 100$ because there are more tokens that connect during one step than the total number of classes. The PROTOGEN$_{(wm)}$'s latency behaves also linearly. It differs from the PROTOGEN$_{(hc)}$'s one in that its increased protection incurs the supplementary cost of sending each class to several tokens in order to guarantee the desired detection probability (in the measures, the minimal detection probability is set to $\texttt{Pr}_{min}(\text{detect}) = 0.99$).

As a conclusion, it appears from these figures that the latency of the PROTOGEN$_{(hc)}$ and PROTOGEN$_{(wm)}$ protocols is primarily determined by the latency of their collection phase, itself being related to the size of the sample of interest in the complete population of tokens.

4.5.4 Detection Probability

Figure 4.7 plots the detection probability with respect to the minimal number of tokens receiving each class for different values of $\texttt{Pr}_{t_2}$. Unsurprisingly, curves are logarithmic; the lower $\texttt{Pr}_{t_2}$, the lower the minimal number of tokens required to reach a desired detection probability.

4.6 Unbreakable Tokens?

As explained in Chapter 3, tokens enforce the highest hardware and software security standards. Actually, this high level of security is a *preventive measure* against attacks since it maximizes the cost of a successful attack on a token. Depending on the underlying application, the benefits resulting from breaking the security enclosure of a token *may however cover the cost of a successful attack* leading to a poor Cost/Benefit attack ratio. In this section, we add a *curative measure* to PROTOGEN. The curative measure

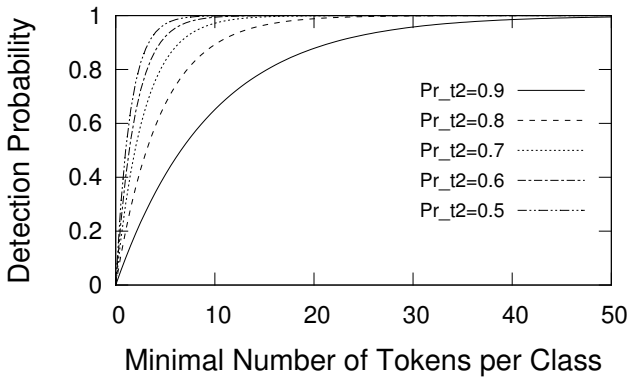


Figure 4.7: Detection Probability of `Invariance` Violation

is in charge of limiting the benefits resulting from breaking one or more tokens, thereby safeguarding the Cost/Benefit ratio.

In the previous PROTOGEN$_{(wm)}$ protocol, if the publisher succeeds in breaking at least one token, it unveils not only the token'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 several keys and to organize the encryption process so that the impact of compromising one key is divided by the number of keys. Consequently, we partition tokens into a set of *clusters*, denoted C, *randomly* and *uniformly*, such that tokens belonging to different clusters are equipped with distinct cryptographic materials. Therefore breaking a token 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 tokens that are members of the broken cluster, but also to encrypt and sign data that originates from the broken cluster: weakly-malicious forge actions now yield tuples that pass the `Origin` property test.

This section first describes an adaptation of the PROTOGEN$_{(wm)}$ protocol to a clustered world. Second, it introduces the curative measure, i.e., a new safety property guaranteeing that weakly-malicious forge actions are harmless, and describes its enforcement. Finally, it integrates the curative measure into PROTOGEN$_{(wm)}$ to yield the PROTOGEN$^{\dagger}_{(wm)}$ protocol.

4.6.1 Clustered tokens

Clustering cryptographic materials limits the decryption ability of tokens to tuples originating from their own clusters. To tackle this limitation, each token θ participating in the **collection** phase embeds the identifier of its cluster, denoted $\theta.C$ and written

CID for short, into the tuples collected. Hence, each token θ participating in the **sanitization** phase is able to ask the publisher to send it a class into which $\theta.C$ appears. However, a side effect of communicating to the publisher the cluster identifier of the token is to reveal directly the cluster identifier of the returned sanitized tuples, based on which the publisher can link subsets of returned decrypted sensitive values to subsets of collected tuples. To avoid this attack, the choice of the downloaded class must be made in stand alone by the token, to which the publisher has previously sent the list of cluster identifiers appearing in every class not fully sanitized yet.

Transferring the complete list of cluster identifiers per class can incur a significant network overhead. This overhead can be reduced by representing the list of cluster identifiers of each class by a bitmap such that for all clusters appearing in the class, the bit at index CID is set to 1. Each list is thus made of n_C bits and there are $n_{\mathcal{E}}$ lists at most (i.e., when no class has been fully sanitized yet): the total overhead amounts to transferring $n_C \times n_{\mathcal{E}}$ bits. At the rate of 8Mbps (i.e., the current effective throughput measured on the hardware platform shown in Fig. 3.1), this does not present any bottleneck. Finding a class into which the token's CID appears has a negligible cost since it consists in checking a single bit per class bitmap.

Special care must also be taken with the number of sanitized tuples returned by a token. Indeed, given a sanitized class within which tuples are grouped by cluster identifier, the returned sensitive values correspond to the group of tuples of the same cardinality. For example, assume that a class contains four tuples from the cluster 1 and six from the cluster 2. When a token downloads the class and returns, e.g., four decrypted sensitive values, the publisher learns that they correspond to the four quasi-identifiers of cluster 1. To avoid this inference, tokens must equalize the number of tuples that they return for each class: whatever their cluster, they must return at most GCD_i tuples, GCD_i being the greatest common divisor of the cardinalities of the subsets of tuples inside the class $\mathcal{E}_i \in \mathcal{E}$. A token downloading the class $\mathcal{E}_i$ easily computes GCD_i based on the cluster identifiers embedded within tuples. Note that the publisher cannot tamper the cluster identifiers: its integrity is protected by the `Origin` safety property.

4.6.2 Defeating Forge Actions

Typicality Tests

Weakly-malicious forge actions reduce the effective k-anonymity of a class by producing a class that contains j collected tuples, with $0 < j < k$, and injecting $(k - j)$ tuples forged, encrypted and signed based on the cryptographic materials of the broken cluster (let assume for the moment that a single cluster is broken). An efficient attack produces

a class that contains far more forged tuples than collected tuples, the extreme case occurring when $j = 1$. 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 tokens are randomly and uniformly partitioned into clusters, the publisher should receive roughly the same number of tuples per cluster, scattered uniformly into the equivalence classes. Thus, inside a class affected by a forge action, all clusters participate roughly with the same number of tuples, except the broken one that participates more than the others.

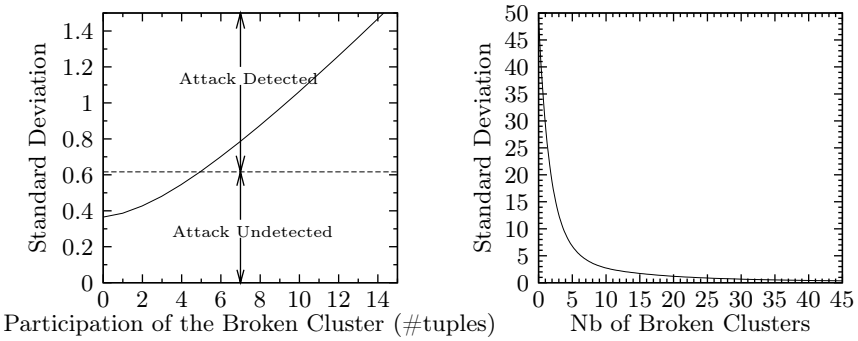
We thus define the `Typicality` safety property based on this observation. `Typicality` (Def. 17) states that the participation of all clusters within a given class must be statistically typical with respect to the participation of the other clusters. The above discussion can be generalized to an arbitrary number of broken clusters. Obviously, the more numerous the broken clusters, the less atypical their participations. However, the high cost of breaching the tamper-resistance of even a single token makes unrealistic the breaching of numerous clusters.

Definition 17 (`Typicality`)**.** Let $\mathcal{E}_j.P$ denote the set of clusters participations in the class $\mathcal{E}_i \in \mathcal{E}$, and `T` denotes the statistical typicality test used by tokens (e.g., a standard deviation analysis). The class $\mathcal{E}_i$ respects the `Typicality` safety property if $\texttt{T}(\mathcal{E}_i.P) = 1$.

Tokens enforce the `Typicality` property at the reception of a class, by analyzing statistical properties of the participation of clusters within the class. We demonstrate experimentally below 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 (e.g., measures used for outliers detection [Barnett94]), the standard deviation analysis already reaches high detection rates despite its simplicity.

Detection Probability

We consider a population under study of $n_\Theta = 1$ million tokens randomly partitioned in $n_C = 500$ clusters; all tokens participate to the collection phase. In our experiments, all clusters are of equal size, but comparable results would be achieved with a normal distribution of tokens into clusters. We implemented the MONDRIAN generalization algorithm (see Section 2.2, which divided the dataset into 8000 classes containing at least $k = 100$ tuples each. Increasing the size of the population yields similar results in terms of detection. To test the typicality of a cluster participating in the class, we compute σ the standard deviation of clusters participations (excluding non-participating clusters). In the general case, where $n_C \geq k$ and there are no tuples



(a) To the Participation of the Broken Cluster (b) To the Number of Broken Clusters

Figure 4.8: Standard Deviation Sensitivity

forged, σ is very small (in our experiments, its average value was $\sigma_{avg} \approx 0.36$ and its largest observed value was $\sigma_{max} \approx 0.62$).

Figure 4.8(a) shows the evolution of σ function of the number of tuples forged by a publisher having broken a single token (then a single cluster). In order to achieve perfect knowledge of a target tuple (e.g., the tuple of a target individual identified by its quasi-identifier value), the publisher would need to forge and inject $k-1$ tuples (in our example, 99 tuples) in the class of the target tuple. As shown in Figure 4.8(a), a cluster participating more than 5 tuples leads to a statistically improbable value (i.e., $\sigma > \sigma_{max}$). Note that Figure 4.8(a) is a zoom: evolution is not linear but polynomial. If the publisher succeeds in breaking several clusters (meaning breaking tokens from different clusters), fake tuples have less impact on the standard deviation because the publisher can distribute the participation over them. Figure 4.8(b) illustrates the value of σ function of the number of broken clusters over which the publisher distributed uniformly $k-1=99$ forged tuples: 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, the standard deviations (average and maximal) are obviously inversely proportional to the number of clusters.

Although more complex statistical analysis could be used (e.g., combining several statistical measures, using outliers detection techniques [Barnett94]), the above experimental results show that even a simple analysis of the standard deviation already makes forge actions harmless.

4.6.3 The Protogen$^{\dagger}_{(wm)}$ Protocol

The additions of PROTOGEN$^{\dagger}_{(wm)}$ to the sanitization phase performed by tokens in the PROTOGEN$_{(wm)}$ protocol are described in Algorithm 3 (for the reading clarity, we only write the PROTOGEN$^{\dagger}_{(wm)}$ specific elements).

Algorithm 3: Sanitization Phase of PROTOGEN$^{\dagger}_{(wm)}$ - Token Side

input : The input of the PROTOGEN$_{(wm)}$ protocol plus: the token's cluster identifier $\theta.C$, its bitmap representation $\beta_{\theta.C}$, the statistical typicality test `T` (where `T` inputs a set of natural numbers and outputs a Boolean).

output: A set of decrypted sensitive values or nothing.

1 Receive the bitmaps B of cluster identifiers, where $\beta_i \in B$ is the bitmap representation of the cluster identifiers appearing in the class $\mathcal{E}_i \in \mathcal{E}$;

2 Choose a class $\mathcal{E}_i$ not sanitized yet such that $\beta_i \wedge \beta_{\theta.C} \neq 0$, where $\wedge$ denotes the binary AND;

3 Download the tuples $\mathcal{E}_i.T$;

4 Compute the set of clusters participations in the class: $\mathcal{E}_i.P$ and check that $\mathtt{T}(\mathcal{E}_i.P) = 1$;

5 Compute the greatest common divisor of participations GCD_i;

6 **return** *The decrypted sensitive values of* GCD_i *tuples randomly chosen and which cluster identifier is* $\theta.C$*;*

4.7 Synthesis

Synthesis. The PROTOGEN suite has allowed us to validate the approach promoted in this thesis. The honest-but-curious publisher attack model made us define the PROTOGEN's execution sequence by remodeling the internals of the traditional collection, construction, and sanitization PPDP phases, and specifying the information voluntarily disclosed to the publisher in order to permit its participation in the construction phase. Then, we shielded the execution sequence against any attack possible that a weakly-malicious publisher could conduct. To this end, we precluded exhaustively the tampering actions upon which weakly-malicious attacks are based, letting tokens assert a small set of safety properties. Finally, we considered the case of a weakly-malicious attacker able to break at least one token, and updated the execution sequence to limit the scope of such a breach.

Towards Meta-Protocol. We based the PROTOGEN suite of protocols on two

simplifying assumptions: (1) we restricted the study to generalization-based algorithms, and (2) we limited each phase to a single execution step in order to disallow - by design - attacks on the execution sequence integrity. Although they proved to be convenient for clearing the problem, time has come to unleash our study from these assumptions. First, both the processing model of the sanitization phase and the safety properties are intricated with the notion of equivalence class. The full **Genericity** objective of the thesis cannot be achieved without clearly separating the internals of the protocol from any algorithm-dependent specificity. Second, the "single execution step" constraint forces the implementation of safety properties to consist of a single execution step. This precludes tokens to communicate during the sanitization phase (which would require at least two steps, i.e., one for sending and another one for receiving). This constraint primarily impacts the `Safe Rules` property requiring it to be verifiable only based on the unitary data structure received (i.e., in PROTOGEN, the equivalence class). Although this is true for the k-ANONYMITY privacy model, this is not necessarily true for other models (e.g., the (d, γ)-PRIVACY model). This constraint also incurs redundant checks of the `Mutual Exclusion` safety property (all tokens participating in the sanitization phase check it), and probabilistic security guarantees strictly lower than 1 with respect to the enforcement of `Invariance`. The META-PROTOCOL suite, described in the next chapter, leans on the study performed for PROTOGEN, freed from its limits.

Chapter 5

The Meta-Protocol Suite of Protocols

Summary: *This chapter describes the* Meta-Protocol *suite of protocols which is the result of abstracting the key findings of the previous chapter. Though the* Meta-Protocol*'s execution sequence is still made of the* ***collection****,* ***construction****, and* ***sanitization*** *phases, the data exchanged as well as the operations performed result from a deep re-modeling of the* Protogen*'s components with the objective to make them unrelated to any specific family of privacy-preserving data publishing algorithm. The safety properties in charge of protecting the* Meta-Protocol *against passive and active attacks include notably the* Protogen*'s safety properties, dissociated from generalization-based algorithms. The contribution of this chapter is thus the achievement of the objective stated in the introduction of the thesis: a decentralized, general, and generic protocol for performing privacy-preserving data publishing algorithms in the* Asymmetric *architecture, namely, the* Meta-Protocol*. The ongoing work presented in this chapter has not been submitted yet.*

Contents

5.1 Introduction

The release of a dataset to various recipients (as planned by the UK and French EHR systems) favours the achievement of numerous and distinct statistical analysis. However, different recipients possibly have different practices, different utility requirements, and are afforded different trust levels. Said in other words: to different recipients, different releases, possibly obtained through different privacy models and algorithms. In our context, the design of a protocol that is as far as possible agnostic with respect to privacy models and algorithms, in short a *meta-protocol*, is consequently decisive.

The Protogen suite presented in the previous chapter has been a step necessary in the achievement of this meta-protocol. Focused on generalization-based privacy algorithms, it identified the main issues posed by the enforcement of privacy-preserving data publishing algorithms within the Asymmetric architecture. The current chapter generalizes the previous chapter, abstracting its key findings in order to achieve the Meta-Protocol suite. The resulting protocols are abstract receptacles that have to be embodied through concrete privacy models and algorithms. We illustrate their genericity through the detailed instantiations of generalization-based algorithms (they are well-known and have been shown to be able to enforce a wide range of privacy models [Machanavajjhala09]) and of the $\alpha\beta$-algorithm [Rastogi07] (it is simple and enforces a variant of Differential Privacy, the strongest widely known privacy model). The Meta-Protocol smoothly adapts to these two types of algorithm despite their substantial differences. Table 5.2 additionally overviews how the Meta-Protocol can support a sample of well-known privacy models.

This chapter is organized as follows. First, Meta-Protocol$_{(hc)}$ tackles the honest-but-curious attack model, defining the shape of the Meta-Protocol suite by abstracting the Protogen's **collection**, **construction**, and **sanitization** phases. We specify the properties that an algorithm should respect to be instantiable under the Meta-Protocol suite, illustrating its high genericity through various well-known privacy models and algorithms. We also show that Meta-Protocol$_{(hc)}$ permits correct and secure instantiations. We then shield it against weakly-malicious actions by integrating a small set of safety properties into its execution sequence. This yields the Meta-Protocol$_{(wm)}$ protocol, also shown to permit correct and secure instantiations. Next, we propose proof-of-concept implementations of the safety properties defined in the first sections. We follow by validating experimentally the approach, focusing on its scalability. Finally, we synthesize the chapter and discuss the Meta-Protocol suite. The interested reader will find the algorithms detailing the complete execution sequences of the $\alpha\beta$-algorithm instantiation in Appendix A.

Symbol	Meaning
$\mathcal{C}$	Construction Information
$\mathcal{P}$	Partitioned Dataset
$n_{\mathcal{P}}$	Number of Tuples in the Partitioned Dataset
$\mathtt{M}^\sharp$	Deterministic MAC Scheme
$\mathtt{P}$	Partitioning Action
$\biguplus$	Embedding Action

Table 5.1: Notations for Meta-Protocol

5.2 Honest-but-Curious Publisher

This section focuses on an honest-but-curious publisher: as previously, the honest-but-curious publisher infers anything feasible to infer but does not deviate from the protocol. We reformulate the **collection**, **construction**, and **sanitization** phases, abstracting the format of collected tuples, the **construction** of sanitization rules, and the sanitization action in order to reach the highest genericity level for the honest-but-curious publisher. Similarly to the Protogen suite of protocols, the Meta-Protocol tackling the honest-but-curious publisher paves the ground on which stronger attackers are tackled. We illustrate its genericity based on the $\alpha\beta$-Algorithm and the generalization-based algorithms instantiations. Their respective execution sequences are depicted in Figures 5.1 and 5.2.

5.2.1 Collection Phase

Similarly to the Protogen's **collection** phase, the Meta-Protocol's **collection** phase is in charge of collecting the dataset to sanitize. Hence, the first information to be contained in a collected tuple is the individual's complete record, encrypted. The tuples collected must also enable the participation of the publisher in the **construction** phase. To this end, they embed a controlled amount of information, based on which the publisher will perform the **construction** phase. The nature of this information depends on the sanitization rules to be constructed by the publisher, themselves depending on the actual privacy algorithm to perform. For example, unveiling the records's quasi-identifier values permits the publisher to construct equivalence classes based on generalization-based algorithms, but obviously does not allow the publisher to perform the $\alpha\beta$-Algorithm. Finally, functional and security information enrich the collected tuples; these auxiliary informations are necessary for the correctness and security properties of algorithms instantiated under Meta-Protocol.

Meta-Protocol$_{(hc)}$

A collected tuple t is thus made of three parts:

- First, the *obfuscated part*, $t.o$, is the result of encrypting the plaintext record of user i: $t.o \leftarrow \mathtt{E}_\kappa(d)$, where d is the raw record, and $\mathtt{E}$ is a secret key encryption scheme semantically secure for multiple messages parametrized by κ, the crypto-material shared among tokens.
- Next comes the *construction part*, denoted $t.c$; it represents the information sent to the publisher in order to enable its participation in the construction of sanitization rules: (1) it must be sufficient for constructing the sanitization rules, while (2) its disclosure must not thwart the privacy guarantees of the model. We keep general the definition of the construction part in order to preserve its adaptability to various algorithms.
- Finally, the *auxiliary part* $t.a$ contains additional information: functional information permitting the distributed execution of the privacy algorithm (their precise definition is also algorithm-dependent), and security information used for protecting the Meta-Protocol from attacks (we will detail them when describing attacks).

As a result, any tuple t has the following form: (o, c, a). At the end of the **collection** phase, the publisher has the *obfuscated dataset* and the *construction information* data structures. The obfuscated dataset, denoted $\mathcal{D}_o$, is made of the obfuscated and auxiliary parts: $\mathcal{D}_o \leftarrow \cup\,(t.o, t.a)$; the *construction information*, denoted $\mathcal{C}$, is made of the construction parts: $\mathcal{C} \leftarrow \cup\,(t.c)$. The information voluntarily disclosed to the publisher (see Definition 9) precisely consists in the construction and auxiliary parts: $\Delta \leftarrow \cup\,(t.c, t.a)$.

Let us illustrate how these data structures support the $\alpha\beta$-Algorithm and the generalization-based algorithms.

$\alpha\beta$**-Algorithm**$_{(hc)}$

During the **collection** phase, each token that connects sends to the publisher one or more tuples, where each tuple follows the standard form: $t \leftarrow (o, c, a)$.

- The obfuscated part $t.o$ is the same as in Meta-Protocol$_{(hc)}$: $t.o \leftarrow \mathtt{E}_\kappa(d)$ where $\mathtt{E}$ and κ are as defined above.

- The construction part $t.c$ must allow the removal of randomly generated tuples that are already in the dataset. Hence, the *equality relationship* between tuples is the necessary and sufficient information to be disclosed to the publisher. Note that since the $\alpha\beta$-ALGORITHM considers that each tuple in $\mathcal{D}$ is unique (see Section 2.2) disclosing the equality relationship does not lead to statistical attacks. We define the construction part as follows: $t.c \leftarrow \mathtt{M}^{\sharp}_{\kappa}(d)$ where $\mathtt{M}^{\sharp} : \{0,1\}^* \rightarrow \{0,1\}^n$ is a *deterministic* MAC scheme (i.e., if $x = y$ then $\mathtt{M}^{\sharp}_{\kappa}(x) = \mathtt{M}^{\sharp}_{\kappa}(y)$), and κ is the token's crypto-material. The deterministic MAC scheme answers the utility constraint of the construction part in that it preserves equality; it answers as well the security constraint in that its outputs are indistinguishable from random bitstrings and can only be computed by parties having the required crypto-material. We refer the interested reader to Section 2.3 for background knowledge about MAC schemes.

- Finally, the functional information of the auxiliary part must allow the tokens in charge of the sanitization to distinguish between tuples from the obfuscated dataset and tuples from the sanitization rules: it is a mere label, e.g., a string of characters, set to "$\mathcal{D}_o$" for tuples from $\mathcal{D}_o$, and denoted $t.a.label$. For now, the security information remains empty.

As a result, the obfuscated dataset is $\mathcal{D}_o \leftarrow \cup\ (\mathtt{E}_{\kappa}(d),$ "$\mathcal{D}_o$"$)$, and the construction information is $\mathcal{C} \leftarrow \cup\ (\mathtt{M}^{\sharp}_{\kappa}(d))$.

Generalization-Based Algorithms

Tuples collected in the context of a generalization-based algorithm for enforcing k-ANONYMITY follow the same abstract form $t \leftarrow (o, c, a)$ but the actual construction and auxiliary parts contain information different from its $\alpha\beta$-ALGORITHM counterpart:

- The construction part $t.c$ must allow the publisher to compute equivalence classes. Disclosing the quasi-identifier values in the clear is sufficient for this, though, depending on the specific algorithm, it may not be necessary. For example, the MONDRIAN algorithm [LeFevre06] (see Section 2.2) needs only the order-relationship between quasi-identifier values, so could be performed by disclosing only the attribute-wise encryption of quasi-identifier values by an *order-preserving encryption scheme* [Agrawal04, Boldyreva09]. However, since k-ANONYMITY does not suffer this disclosure, we define the construction part as follows: $t.c \leftarrow d[QI]$ where $d[QI]$ denotes the quasi-identifier value of the record d. Disclosing the quasi-identifier values answers the utility constraint in that it makes

the publisher able to construct equivalence classes, and answers as well the security constraint in that it does not thwart the k-ANONYMITY privacy model.

- The auxiliary part is empty (no functional information is needed and the security information will be set when considering attacks).

The obfuscated part $t.o$ is the same as in META-PROTOCOL$_{(hc)}$: $t.o \leftarrow \mathtt{E}_\kappa(d)$ where $\mathtt{E}$ and κ are as defined above.

At the end of the **collection** phase, the obfuscated dataset is thus $\mathcal{D}_o \leftarrow \cup\,(\mathtt{E}_\kappa(d), \varnothing)$, and the construction information is $\mathcal{C} \leftarrow \cup\,(d[QI])$.

5.2.2 Construction Phase

Similarly to traditional statistical surveys, the publisher stops the **collection** phase and launches the **construction** phase when it has collected enough data with respect to his needs (i.e., $n_\mathcal{D}$ tuples).

Meta-Protocol$_{(hc)}$

The **construction** phase consists in forming the sanitization rules $\mathcal{R}$ based on the construction information $\mathcal{C}$. The function that inputs $\mathcal{C}$ and outputs $\mathcal{R}$ is algorithm-dependent.

$\alpha\beta$-Algorithm$_{(hc)}$

The rules of the $\alpha\beta$-ALGORITHM are records randomly generated from the domain of all possible tuples *dom*. The publisher, unable to generate tuples that follow the standard form (o, c, a) (it does not have the tokens's crypto-material), relies on tokens to generate tuples. The **construction** phase consists thus, for each token that connects, in generating random records from *dom* and sending $t \leftarrow (\mathtt{E}_\kappa(d), \mathtt{M}^\sharp_\kappa(d), \text{“}\mathcal{R}\text{”})$ for each record d. If the publisher did not receive any tuple (generated or collected) associated to $\mathtt{M}^\sharp_\kappa(d)$ (i.e., $\mathtt{M}^\sharp_\kappa(d) \notin \mathcal{C}$), it keeps t by storing its obfuscated and auxiliary parts in the rules data structure: $\mathcal{R} \leftarrow \mathcal{R} \cup (\mathtt{E}_\kappa(d), \text{“}\mathcal{R}\text{”})$, and its construction part into the construction information data structure: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathtt{M}^\sharp_\kappa(d)$. When the publisher has gathered $n_\mathcal{R}$ random tuples, it stops the **construction** phase and launches the **sanitization** phase.

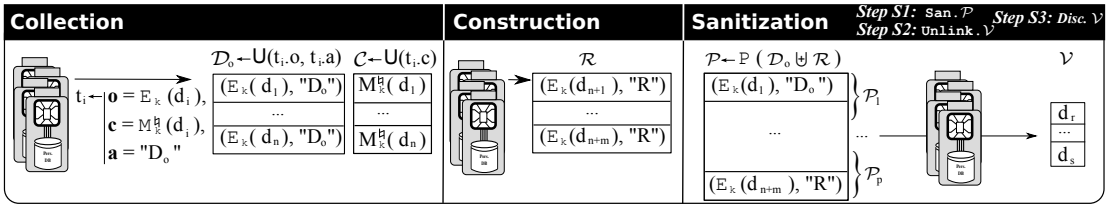


Figure 5.1: Execution Steps and Data Structures of META-PROTOCOL$_{hc}$ Illustrated by $\alpha\beta$-ALGORITHM$_{hc}$

Generalization-Based Algorithms

The sanitization rules of generalization-based algorithms consist in equivalence classes. The publisher constructs them by launching any generalization-based algorithm (e.g., Mondrian [LeFevre06], Incognito [LeFevre05]) on the collected quasi-identifier values.

5.2.3 Sanitization Phase

Meta-Protocol$_{(hc)}$

The **sanitization** phase is in charge of producing the final sanitized release $\mathcal{V}$ by parallelizing the sanitization action on tokens. To this end, the publisher *embeds* the rules within the obfuscated dataset and *partitions* uniformly the resulting data structure such that a partition (1) contains a subset of tuples and their corresponding sanitization rules, and (2) fits the resources of a token. The resulting data structure is called the *partitioned dataset* and denoted $\mathcal{P}$: $\mathcal{P} \leftarrow \mathtt{P}(\mathcal{D}_o \biguplus \mathcal{R})$, where $\mathtt{P}$ denotes the action of partitioning the dataset (also called the *partitioning action*) and $\biguplus$ the action of embedding the rules within the partitions (also called the *embedding action*). Each token that connects is now able to participate in the **sanitization** phase by downloading a self-contained partition $\mathcal{P}_i \in \mathcal{P}$, and uploading on the publisher the result of decrypting and sanitizing the tuples it contains. The partition's size is fixed before the protocol starts, according to the tokens's resources. The publisher then stores the sanitized tuples into $\mathcal{V}$, which is finally released when all the partitions have been sanitized.

$\alpha\beta$-Algorithm$_{(hc)}$

The rules of the $\alpha\beta$-ALGORITHM$_{(hc)}$ are made of tuples. The embedding action $\biguplus$ thus consists in performing the set-union between $\mathcal{R}$ and $\mathcal{D}_o$. Consequently, the partitioned dataset $\mathcal{P}$ is formed as follows: $\mathcal{P} \leftarrow \mathtt{P}(\mathcal{D}_o \cup \mathcal{R})$, where $\cup$ denotes the set-union. Each token that connects downloads a partition $\mathcal{P}_i \in \mathcal{P}$, and for all tuples $t_j \in \mathcal{P}_i$ returns

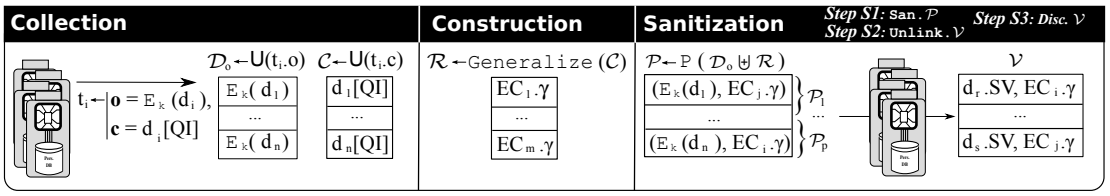


Figure 5.2: Execution Steps and Data Structures of META-PROTOCOL$_{hc}$ Illustrated by Generalization-Based Algorithms

$t_j.o$ decrypted (1) with a probability $\alpha + \beta$ if $t_j.a.label =$ "$\mathcal{D}_o$" and, (2) with certainty otherwise.

Generalization-Based Algorithms

The sanitization rules of generalization-based algorithms are the equivalence classes. The embedding action $\biguplus$ consists in associating to each tuple of the obfuscated dataset the generalization node of the class in which it is contained. The resulting dataset is then partitioned to form $\mathcal{P}$. Each token that connects downloads a partition $\mathcal{P}_i \in \mathcal{P}$, and returns to the publisher the tuple's sensitive value, decrypted, with the associated generalization node.

Passive attacks

The **sanitization** phase described above is not protected against passive attacks from the honest-but-curious publisher. Passive attacks aim at refining the knowledge about raw individual records based on the side-effect information due to partitioning the dataset. Indeed, the partitioning scheme allows the publisher to map (1) each partition to the construction and auxiliary parts of the tuples it contains, and (2) each sanitized record to its corresponding partition. Passive attacks consist in joining these two mappings (modeled as, e.g., two relational tables). For example, with the $\alpha\beta$-ALGORITHM instantiation, assume that a downloaded partition only contains tuples associated to the label "$\mathcal{D}_o$". As soon as the publisher receives the corresponding sanitized records, which are the raw records decrypted, he learns with certainty that they all come from "$\mathcal{D}_o$". Privacy guarantees are jeopardized. Let us consider another example, related this time to the generalization-based algorithms instantiation, and assuming that a downloaded partition only contains tuples associated to distinct equivalence classes. As soon as the publisher receives a sanitized record, i.e., a sensitive value decrypted and its corresponding class, it learns with certainty its corresponding quasi-identifier

value: it is the only one that corresponds to both the partition and the given class. Privacy guarantees are again jeopardized.

We make these mappings harmless by requiring the sanitized release to satisfy the `Unlinkability` safety property *before being disclosed* to the publisher.

Definition 18 (`Unlinkability`)**.** Let Ψ_1 denote the data structure mapping each partition to the construction and auxiliary part of the tuples it contains, and Ψ_2 the data structure mapping each sanitized record to its corresponding partition. Let $\leftrightarrow$ be the binary operator denoting the possible mapping between a sanitized record and (a subset of) construction and auxiliary parts. The sanitized release respects the `Unlinkability` safety property iff

$$\forall v \in \mathcal{V}\ \forall \delta \subset \Delta, \mathtt{Pr}[v \leftrightarrow \delta | \Psi_1, \Psi_2] = \mathtt{Pr}[v \leftrightarrow \delta]$$

Informally speaking, `Unlinkability` asserts that the mapping between each sanitized record and any subset of construction and auxiliary parts is as probable with and without the side-effect information due to the partitioning scheme. See Section 5.4 for a proof-of-concept implementation.

5.2.4 Instantiability under the Meta-Protocol

The Meta-Protocol contains a few actions whose precise semantics depend on the algorithm to be instantiated: (1) the construction of rules, (2) the embedding action, and (3) the sanitization action now parallelized at the partition granularity. Any privacy algorithm can be instantiated under Meta-Protocol provided that the algorithm-dependent actions can be adapted *securely* (i.e., the information voluntarily disclosed does not thwart the privacy guarantees) and *functionally* (i.e., the embedding, partitioning, and sanitization actions are possible, and the information voluntarily disclosed is sufficient to both the construction of rules and the sanitization action). These two requirements are called respectively the *security requirement* and the *functional requirement*.

Table 5.2 gives a concrete sample of models that can be enforced by algorithms instantiable under the Meta-Protocol. For each model and algorithm, we outline the main algorithm-dependent information and operations (i.e., the construction part, the rules, the embedding action, and the sanitization action). We do not aim here at listing exhaustively the models and algorithms supported by Meta-Protocol but rather to show their variety. We briefly recall below the underlying principles of the algorithms mentioned in the table; for more details, the reader is referred to Section 2.2.

Model	Algorithm	$\mathcal{C}$ part	$\mathcal{R}$	Embed. Act.	San. Act.
(d,γ)-PRIVACY	$\alpha\beta$-ALGORITHM	Records's equality relationship	Random records	Set-union between $\mathcal{R}$ and $\mathcal{D}_o$	Sample $\mathcal{D}$ Merge it with $\mathcal{R}$
k-ANONYMITY	Generalization-based algo.	QI	EC.γ	For each tuple, embed its corresponding "EC.γ"	Association EC.$\gamma \rightarrow$ {SV}
idem	MONDRIAN	QI's order relationship	*idem*	*idem*	*idem*
l-DIVERSITY	MONDRIAN	- SV - QI's order relationship	*idem*	*idem*	Association EC.$\gamma \rightarrow$ {SV}
idem	BUCKETIZATION	SV	*idem*	*idem*	Association EC.$\gamma \rightarrow$ {QI}
t-CLOSENESS	SABRE	SV	*idem*	*idem*	Association EC.$\gamma \rightarrow$ {QI}
ϵ-PRIVACY	MONDRIAN	- SV - QI's order relationship	*idem*	*idem*	Association EC.$\gamma \rightarrow$ {SV}
idem	BUCKETIZATION	SV	*idem*	*idem*	Association EC.$\gamma \rightarrow$ {QI}

Table 5.2: A Sample of Supported Models and Algorithms

The models enforced by the BUCKETIZATION [Xiao06] and the generalization-based algorithms (e.g., [LeFevre05, LeFevre06]) consider that records are made of a *quasi-identifier* (*QI* for short) and a *sensitive value* (*SV* for short), based on which records are classified into *equivalence classes* (*EC* for short). Their privacy guarantees are reached by requiring each class to generalize a minimal number of quasi-identifier values, and possibly to exhibit a certain distribution of sensitive values.

Let us briefly recall the basic principles of these algorithms in a trusted publisher context. Generalization-based algorithms form a set of equivalence classes, where in each class, all tuples share the same generalization node. If required by the model, the formation of classes takes into account the distribution of sensitive values within each class. The BUCKETIZATION algorithm also forms a set of equivalence classes but based only on the sensitive values of records; their generalization node is consequently not a coarsened quasi-identifier but an identifier assigned to the subset of records that form the equivalence class. The SABRE algorithm is a bucketization-based algorithm. Note that in Table 5.2, the sanitization action of bucketization-based algorithms could associate to each class a generalized quasi-identifier value rather than a set of quasi-identifier values.

The models enforced by these algorithms encompass notably the k-ANONYMITY

model [Sweeney02] (that do not take into account the sensitive values distribution within each class), and the l-DIVERSITY [Machanavajjhala06] and ϵ-PRIVACY [Machanavajjhala09] models (that additionally do). Hence, the latter can be enforced through both the BUCKETIZATION algorithm and generalization-based algorithms, though the amount of information disclosed in using one or the other differs.

Indeed, using generalization-based algorithms for enforcing models that constrain both the quasi-identifier and the sensitive values requires disclosing information concerning both. Generalizing the quasi-identifier requires, e.g., the *order relationship* between them in order to let the MONDRIAN algorithm [LeFevre06] partition the tuples ordered by their quasi-identifier parts, while constraining the distribution of sensitive values requires the *equality relationship* between sensitive values. Recently, some works have focused on the order-preserving encryption problem [Agrawal04, Boldyreva09]. Encrypting quasi-identifier values based on these schemes permits to disclose the order relationship between quasi-identifier attributes values while keeping them still obfuscated; and providing the sensitive values in the clear straightforwardly discloses the equality relationship between sensitive values. As a result, for enforcing such models, the construction part of each tuple could consist in the order-preserving encryption of the record's quasi-identifier attribute values with its sensitive value in the clear. On the contrary, the BUCKETIZATION algorithm requires only the disclosure of the equality relationship between sensitive values: it does not take into account the quasi-identifier values for producing equivalence classes.

Note that the state-of-the-art centralized publishing privacy algorithms have been primarily designed to fit the trusted publisher context; some of them cannot cope with META-PROTOCOL's instantiability constraints. Nonetheless, this does not mean that the privacy models that they enforce cannot be supported by META-PROTOCOL: instantiable algorithms may be able to enforce them. For example, the META-PROTOCOL cannot use directly the INCOGNITO algorithm [LeFevre05] to enforce the ℓ-DIVERSITY model because it would require sharing both quasi-identifier values and sensitive values in the clear, consequently missing the security requirement. Nevertheless, using the BUCKETIZATION algorithm (based on disclosing sensitive values in the clear only) or MONDRIAN (based on disclosing the quasi-identifier values encrypted by an order-preserving preserving encryption scheme, and the sensitive values in the clear) enables the enforcement of ℓ-DIVERSITY with the META-PROTOCOL. Designing algorithms especially for the ASYMMETRIC architecture is let to future works (see the perspectives drawn in Section 7.3).

5.2.5 Correctness and Security

Any privacy algorithm implemented under Meta-Protocol$_{(hc)}$ is totally correct and secure as stated by theorems 6 and 7.

Theorem 6. Let $\pi_{(hc)}$ be the instantiation of the privacy algorithm $\mathbb{A}$ under Meta-Protocol$_{(hc)}$. Given that $\pi_{(hc)}$ respects the functional requirement, then $\pi_{(hc)}$ is *totally correct*.

Proof. The publisher forms the dataset to sanitize during the **collection** phase. This essentially amounts to sampling the set of tokens similarly to a trusted server context. The **collection** phase stops when the dataset meets user-defined constraints (e.g., a sufficient cardinality). The **construction** phase then starts and produces the sanitization rules based on the construction parts of tuples collected. Provided that the construction part meets the functional requirement, the **construction** phase is similar to its usual centralized counterpart. Finally, the **sanitization** phase makes use of *any* connected token to sanitize the dataset partition by partition. The information contained in each partition is sufficient for its sanitization because a partition includes both the tuples to sanitize and the sanitization rules to be applied. The **sanitization** phase stops when enough tokens have been connected for sanitizing all partitions. The **collection** and **construction** phases are similar to their centralized counterpart; the sanitize phase too because the implementation of `Unlinkability` is required *by definition* to preserve its correctness. As a result, since all the phases are similar to their centralized counterpart, $\pi_{(hc)}$ is *correct*; and because tokens connect indefinitely, $\pi_{(hc)}$ also terminates. □

Theorem 7. Let $\pi_{(hc)}$ be the instantiation of a privacy algorithm $\mathbb{A}$ under Meta-Protocol$_{(hc)}$. Given that $\pi_{(hc)}$ respects the security requirement, then $\pi_{(hc)}$ securely computes $\mathbb{A}$ against a honest-but-curious publisher.

Proof. An honest-but-curious publisher obtains the partitioned dataset $\mathcal{P}$, the construction information $\mathcal{C}$, and the sanitized release $\mathcal{V}$, and analyzes them in order to thwart the expected privacy guarantees. First, let consider $\mathcal{P}$ and $\mathcal{C}$. Under the assumption that semantically secure encryption schemes exist, it is infeasible to deduce any information about a plaintext record by looking at an obfuscated part in $\mathcal{P}$. The remaining information, made of the auxiliary parts in $\mathcal{P}$ and the construction parts in $\mathcal{C}$, does not thwart the privacy guarantees *by definition* (security requirement), and is also considered to be known by the attacker in the ideal setting (as stated in Def. 9). As a result, the knowledge of $\mathcal{P}$ and $\mathcal{C}$ does not bring any information to the adversary in the real setting that its counterpart in the ideal setting does not already have.

Next, let consider the sanitized release $\mathcal{V}$ independently from the other data structures. Whatever the setting, be it real or ideal, the attacker has access to the complete set of partitions in $\mathcal{V}$. The difference between the real and the ideal settings is that in the real setting, the attacker may be able to access a subset of sanitized tuples made unlinkable. Nevertheless, in the ideal setting, the attacker can simulate this access by randomly sampling the sanitized release. Thus, the attacker in the real setting does not learn anything more based on $\mathcal{V}$ than its counterpart in the ideal setting. Finally, any attack based on a joint analysis of the three data structures - $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{V}$ - consists in mapping construction/auxiliary information to their corresponding sanitized tuple. However, the `Unlinkability` safety property disables such mappings. As a result, the adversary in the real setting cannot obtain any information from $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{V}$, that its counterpart in the ideal setting is unable to obtain. □

5.3 Weakly-Malicious Publisher

META-PROTOCOL$_{(hc)}$ copes with an honest-but-curious publisher; this section carries on the dissociation process, focusing now on a weakly-malicious publisher. Furthermore, the weakly-malicious publisher reaches in this section its full potency: in addition to the tampering actions (forge, copy, delete) and to the production of unsafe rules, it can attack the execution sequence of the **sanitization** phase - which is not required anymore to be made of a single execution step.

In this section, we first redefine the safety properties in charge of precluding tampering actions such that they guarantee now the integrity of the partitioned dataset. These safety properties can be used for any dataset. Second, we rethink the safety property in charge of guaranteeing that the sanitization rules enforce the required privacy guarantees given a partitioned dataset. We show third how to guarantee the integrity of the execution sequence. Fourth, META-PROTOCOL$_{(hc)}$ is upgraded by embedding the safety properties into its execution sequence, resulting in META-PROTOCOL$_{(wm)}$. The latter is then shown to enable correct and secure instantiations (provided that the implementation of the safety properties preserve the protocol's correctness and security). Note that this section focuses on abstracting the safety properties (definition stage); their implementation (construction stage) is delayed to Section 5.4.

5.3.1 Partitioned Dataset Integrity

We consider the usual *forge*, *copy*, and *delete* tampering actions (note that, with respect to data authentication, modifying a tuple is equivalent to forging a tuple) and guarantee the integrity of a given dataset by requiring it to assert a small set of safety properties

checked by the tokens. Indeed, the integrity of any dataset, partitioned beforehand, can be guaranteed by these safety properties. Simple examples of attacks based on tampering actions include, e.g., forging all the tuples of the obfuscated dataset except one such that the non-forged sanitized record be easily identified in the sanitized release, or copying a collected tuple a certain number of times and identify in the sanitized release its corresponding sanitized record based on the number of times it appears.

Preclude Forge Actions

The *forge* tampering action allows the attacker to forge and inject tuples into the dataset. The `Origin` safety property, in charge of precluding forge actions, was defined in the previous chapter independently from any algorithm. We recall this definition to make the chapter self-contained. `Origin` states that any data structure originating from a token must be accompanied with its signature as a proof of authenticity.

Definition 19 (`Origin` *(from Chap. 4)*)**.** Let $(b, \sigma) \in \{0,1\}^* \times \{0,1\}^n$ be a bitstring (i.e., any data structure) with its n-bit secret key signature. The data structure b respects the `Origin` property if $\sigma = \mathtt{M}_\kappa(b)$, where $\mathtt{M}_\kappa$ is the message authentication code scheme parametrized by the token's cryptographic material.

Each tuple t embeds a signature, denoted $t.\sigma$, which is the result of signing the tuple's obfuscated and auxiliary parts: $t.\sigma \leftarrow \mathtt{M}_\kappa(t.o||t.a)$, where $||$ denotes the bitwise concatenation.

Preclude Copy Actions

Similarly to copy actions within PROTOGEN$_{(wm)}$, the nature of copy actions within META-PROTOCOL$_{(wm)}$ is twofold. With *intra-partition* copy actions, a tuple (or more) is copied several times into its own partition, while with *inter-partition* copy actions, the destination partition is different from the source partition. The safety properties in charge of precluding intra/inter-partition copy actions are based on (1) assigning to each tuple a unique identifier, and (2) organizing the dataset such that each tuple is authorized to be part of a single partition. In PROTOGEN$_{(wm)}$, the equivalence classes already defined an implicit organization among tuples. In META-PROTOCOL$_{(wm)}$ this organization does not exist natively but has to be constructed; we construct it based on tuple identifiers.

Let us focus first on intra-partition copy actions by assuming that there is a single partition. The `Identifier Unicity` safety property (Def. 20) precludes intra-partition copy actions by requiring that each tuple identifier of the partition be unique.

Definition 20 (`Identifier Unicity`)**.** Let $t \in \mathcal{P}_i$ be a tuple in the partition $\mathcal{P}_i$, and $t.a.\tau$ denote t's identifier. The partition P_i respects the `Identifier Unicity` safety property if for every pair of tuples $t_j, t_k \in \mathcal{P}_i$, $j \neq k \Rightarrow t_j.a.\tau \neq t_k.a.\tau$.

Now, let us consider several partitions. In addition to being unique in its own partition, each tuple must also appear in a single partition. To this end, we define for each partition the set of tuple identifiers it is *supposed* to contain, called the partition's TID-Set, and denote it $\mathcal{P}_i^\tau$, where $\mathcal{P}_i$ is a partition.

First, the `Membership` safety property (Def. 21) states that each tuple must be in the partition it is supposed to belong to. Second, the `Mutual Exclusion` safety property (Def. 22) ensures that each tuple is supposed to belong to a single partition.

Definition 21 (`Membership`)**.** A partition $\mathcal{P}_i$ respects `Membership` if for every tuple $t_j \in \mathcal{P}_i$, then $t_j.a.\tau \in \mathcal{P}_i^\tau$.

Definition 22 (`Mutual Exclusion`)**.** Partitions respect `Mutual Exclusion` if for every pair of partitions $\mathcal{P}_i, \mathcal{P}_j \in \mathcal{P}$, $i \neq j \Rightarrow \mathcal{P}_i^\tau \cap \mathcal{P}_j^\tau = \varnothing$.

We can easily observe that the `Identifier Unicity`, `Membership`, and `Mutual Exclusion` properties together guarantee the absence of intra/inter-partition copy actions.

Theorem 8. Enforcing together the `Identifier Unicity`, `Membership`, and `Mutual Exclusion` safety properties is necessary and sufficient for guaranteeing the absence of any weakly-malicious copy action.

Proof. The proof is similar to its PROTOGEN's counterpart. 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 integrity of a tuple identifier is guaranteed by the safety property in charge of precluding forge-based attacks). Second, assume that a given tuple t has been copied into two distinct partitions. Only the TID-Set of one of them is supposed to contain t's identifier because otherwise `Mutual Exclusion` would be contradicted. Consequently there must be one partition's TID-Set that does not contain t's identifier. This contradicts the `Membership` safety property. As a result, `Membership` and `Mutual Exclusion` are together sufficient for precluding inter-partition copy actions.

We now show the necessity of these properties. First, since a distinct identifier is assigned to each 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 (also because a distinct

identifier is assigned to each tuple) and (2) the `Membership` property too in that each tuple appears in the partition which TID-Set contains its identifier. □

Preclude Delete Actions

A partition is affected by a *delete* tampering action if its TID-Set has been associated to (at least) two distinct sets of tuples. To avoid such actions, the content of each partition must not change along the whole protocol: it must be made *invariant*. Contrary to PROTOGEN$_{(wm)}$, the META-PROTOCOL$_{(wm)}$'s **sanitization** phase is allowed to consist of several execution steps, permitting the definition of a non probabilistic (so stronger) `Invariance` (see Section 5.4 for its implementation) and the complete achievement of the desired secure computation definition.

Definition 23 (`Invariance`)**.** Let $LABEL$ be a set of character strings containing the labels designating the data structures to be made invariant; $LABEL$ is known by both the publisher and the tokens. Let $l_0 \in LABEL$ 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 token 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 token, 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 partitioned dataset is invariant if the couple ("$\mathcal{P}$", $\mathcal{P}$) respects the `Invariance` safety property, "$\mathcal{P}$" being the partitioned dataset's label and $\mathcal{P}$ its actual bitstring representation.

5.3.2 Preclude Unsafe Rules

The attacker could endanger the privacy guarantees by producing rules inconsistent with respect to the dataset: they must be checked before being applied. The nature of rules varies highly from one algorithm to the other (e.g., false tuples for the $\alpha\beta$-ALGORITHM, equivalence classes for generalization-based algorithms) thereby precluding any generic checking protocol. However, most rules (if not all) share a common feature: their privacy guarantees are based on the actual *number of some algorithm-dependent data* (e.g., [Sweeney02, Machanavajjhala06, Li10b, Nergiz07b, Chen07, Martin07, Rastogi07, Machanavajjhala09] and see Section 2.2). For example, the privacy guarantees enforced by the $\alpha\beta$-ALGORITHM for the (d,γ)-PRIVACY model are based on the number of tuples from $\mathcal{D}_o$ and the number of those from $\mathcal{R}$, k-ANONYMITY [Sweeney02] is based on the number of tuples in each equivalence class, l-DIVERSITY [Machanavajjhala06] is based on the frequencies of sensitive values in each equivalence class. This recurrent need motivates the design of a secure counting protocol between tokens and the

publisher; we denote it `Count`. `Count` is a generic algorithm (see Section 5.4.3 for a proof-of-concept implementation); when used for checking the rules, `Count` inputs the partitioned dataset $\mathcal{P}$ and outputs the correct corresponding set of counts. Note that counts related to rules are not private; they were already known to the (untrusted) publisher to build rules respecting the privacy guarantees. The safety property corresponding to checking the rules is called `Safe Rules`. We omit its formal definition because it would only present a limited interest: first, it would be abstract, having to be declined for each privacy model; second, it would simply reflect the privacy guarantees of each model.

5.3.3 Integrity of the Execution Sequence

Meta-Protocol$_{(wm)}$ is made of a sequence of steps beginning by the collection of tuples, going through calls to safety properties, and finishing by the disclosure of sanitized partitions. Checking the completeness and order of the execution sequence is critical (e.g., the sanitization of partitions 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 publisher essentially aims at obtaining individual records: he necessarily executes the first step - i.e., the initial 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, tokens embed the *expected execution sequence* of the algorithm instantiated under Meta-Protocol, 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 publisher 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 token connects, the publisher sends 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 token 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). See Section 5.4 for the implementation of proofs.

Definition 24 (`Execution Sequence Integrity`)**.** Let S be the pre-installed finite set of execution steps of the current algorithm's execution sequence. Let $s \in S$ denote the current execution step sent to the connecting token by the publisher, and $s.P$ denote

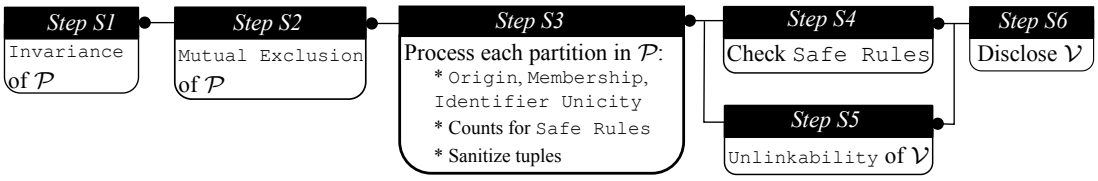


Figure 5.3: Execution Steps of META-PROTOCOL$_{(wm)}$'s **Sanitization** Phase

the set of proofs required for executing s as specified in S. Finally, let P_{pub} denote the set of proofs received from the publisher. The `Execution Sequence Integrity` safety property is respected iff: $\forall\, s \in S,\ \forall\, p \in s.P$ then $\exists\, p_{pub} \in P_{pub}$ s.t. $\texttt{VALID}(p, p_{pub}) = \texttt{TRUE}$, where `VALID` checks the validity of the proofs provided by the publisher (its precise specification depends the proofs implementations at each step - see Section 5.4).

5.3.4 Integrating the Safety Properties into the Execution Sequence

Figure 5.3 depicts the execution sequence of META-PROTOCOL$_{(wm)}$'s **sanitization** phase. The execution flows from step S0 to step S6, the arrows beginning with a dot denoting a precedence relationship between two steps (concretized by a check of the `Execution Sequence Integrity` safety property).

The execution order of the sequence is dictated by two observations:

- The partitioned dataset must be protected against tampering actions before producing the `Counts`, that have to be then input by the `Safe Rules` property.
- The `Unlinkability` safety property must be enforced only once tuples are sanitized. To understand why, assume that we made $\mathcal{P}$ unlinkable instead of $\mathcal{V}$: tuples in each partition would be fully obfuscated and could not be linked to any specific subset of construction or auxiliary parts. However, when obtaining a sanitized partition, the publisher could learn a certain amount of information about the subset of tuples and rules embedded in the partition. For example, considering the $\alpha\beta$-ALGORITHM, by analyzing the number of tuples dropped in a sanitized partition the publisher deduces the approximate number of tuples from $\mathcal{D}_o$ and $\mathcal{R}$. Enforcing the `Unlinkability` safety property on sanitized tuples guarantees that no side-effect information is unveiled when the sanitized release is disclosed.

Note that in Figure 5.3 the Step S1 (`Invariance`) precedes the Step S2 (`Mutual Exclusion`). According to this order, the `Invariance`'s proof is checked during Step S2 by the token asserting `Mutual Exclusion`. Consequently, the tokens performing

Step S3 only need to check the `Mutual Exclusion`'s proof to know that both `Mutual Exclusion` and `Invariance` have been previously checked. There is no strong need in making Step S1 precede Step S2; they could be executed in parallel provided that each token performing Step S3 asserts that both `Invariance` and `Mutual Exclusion` have been checked previously.

The Meta-Protocol$_{(wm)}$'s execution sequence contrasts with the Protogen$_{(wm)}$'s sequence on the number of execution steps. Why is it made of five more steps? First cause: the impact of the abstraction process. Indeed, in Protogen, the unitary data structure processed by token is the equivalence class: tuples appearing in the same class are in the same unitary data structure. On the contrary, the generic unit processed by tokens in Meta-Protocol is the partition. Additional steps are thus necessary for disabling inferences through the `Unlinkability` property, and for permitting a generic check of the `Safe Rules` that takes into account tuples present in the other partitions. Second cause: the `Invariance` safety property is certain and not probabilistic anymore, and the `Mutual Exclusion` is checked once and for all. This introduces additional steps for letting tokens communicate about the invariant and the mutually exclusive states of the dataset.

We illustrate the full Meta-Protocol$_{(wm)}$'s execution sequence by instantiating it under the $\alpha\beta$-Algorithm$_{(wm)}$. The interested reader will find in Appendix A the detailed execution sequence embedding the safety properties's implementations.

$\alpha\beta$-Algorithm: Before the Sanitization Phase

Collection Phase. The **collection** phase is similar to its counterpart in the honest-but-curious version of the protocol. Tuples collected are in the form $(\mathtt{E}_\kappa(d), \mathtt{M}^\sharp_\kappa(d), a, \sigma)$, where $t.a$ contains (1) $t.a.\tau = \mathtt{M}^\sharp_\kappa(d)$ (recall that the $\alpha\beta$-Algorithm assumes that records are unique whether collected or randomly generated, a collision on identifiers is thus highly improbable - see Section 5.4), (2) $t.a.label =$ "$\mathcal{D}_o$" (as previously).

Construction Phase. The **construction** phase requires a slight adaptation for coping with a weakly-malicious adverary. In the honest-but-curious version of the protocol, the publisher is in charge of deleting the false tuples that are duplicates of true ones. A weakly-malicious publisher may deviate from this protocol by simply doing the reverse: deleting the true tuples and keeping the false ones. The insertion of the latter into $\mathcal{V}$ will thus occur with certainty rather than probabilistically (the expected behavior): the $\alpha\beta$-Algorithm is flawed, its privacy guarantees thwarted. The publisher could further build on this breach by waiting for the receival of a set of false tuples that cover a full subset of the true tuples.

We preclude such attack by hiding to the publisher the equality relationship between

true and false tuples while still guaranteeing the absence of duplicates in $\mathcal{V}$: tokens will directly generate false tuples from $(dom - \mathcal{D})$. Let us intuitively describe the counter-measure. As previously, the publisher starts the **construction** phase after having collected the true tuples. A (sketchy) naive way to allow each connecting token to generating false tuples from $(dom - \mathcal{D})$ could consist in sending it the complete list of MACs: the token would randomly generate a false record and check that its MAC is absent from the list received. Though this naive solution closes the above breach, it suffers in practice from a high transfer cost (the number of MACs transfered is equal to the initial dataset cardinality). We propose to control (and reduce) the transfer cost by *summarizing* the list by a set of intervals where each interval covers several MACs. Intervals are built by scanning the sorted list of MACs, replacing each *dense* area of the list by a covering interval. A density threshold allows to control the number of intervals and consequently the transfer cost. Connecting tokens simply download the set of intervals and generate false tuples which associated MACs do not appear into the dense intervals. Tuples randomly generated follow the same form as tuples collected but differ (1) in their label, set to "$\mathcal{R}$" instead of "$\mathcal{D}_o$" and (2) in their deterministic MAC, computed based on another key (i.e., in order to hide to the publisher the equality relationship with true tuples). Note that in order to force the publisher to produce a single set of intervals consistent with the initial dataset, intervals have to be made invariant and their consistency checked when sanitizing records (i.e., any true record belongs to a dense area and any false record to a sparse area). We do not describe this counter-measure further - it would only present a limited interest here.

$\alpha\beta$-Algorithm: Sanitization Phase

The partitioned dataset is formed as previously. First safety property to be enforced, the `Invariance` (step S1) allows the other safety properties related to the partitioned dataset to sit on a stable basis (in addition to precluding delete actions). Step S2 checks `Mutual Exclusion`. The partitioned dataset is then downloaded partition per partition at step S3, during which the partition-wise safety properties are checked (i.e., `Origin`, `Identifier Unicity`, `Membership`), the algorithm-dependent `Counts` are formed (i.e., number of tuples from "$\mathcal{D}_o$" and from "$\mathcal{R}$" in the partition), and the records are sanitized. Step S3 also checks that `Mutual Exclusion` was asserted previously. Sanitized records are not disclosed immediately to the publisher: the safety of rules is checked based on the counts performed in step S3 (i.e., the actual total numbers of "$\mathcal{D}_o$" and "`R`" tuples must be as expected), and the `Unlinkability` of the sanitized release is enforced (see Section 5.4 for the implementation). Steps S4 and S5 respectively handle these properties. Finally, the sanitized release is disclosed to the publisher in step S6.

5.3.5 Correctness and Security

Theorems 9 and 10 state that any privacy algorithm implemented under META-PROTOCOL$_{(wm)}$ is totally correct and secure.

Theorem 9. Let $\pi_{(wm)}$ be the instantiation of the privacy algorithm $\mathbb{A}$ under META-PROTOCOL$_{(wm)}$. Given that $\pi_{(wm)}$ respects the functional requirement, then $\pi_{(wm)}$ is totally correct.

Proof. Tuples collected for META-PROTOCOL$_{(wm)}$ are formatted identically to those collected for META-PROTOCOL$_{(hc)}$. The former differ from the latter in that they contain additional information: identifiers and signatures of tuples. Hence, the correctness of the META-PROTOCOL$_{(wm)}$'s **collection** and **construction** phases is guaranteed if the correctness of their META-PROTOCOL$_{(hc)}$'s counterpart is guaranteed. As stated in Theorem 6, this is the case. The META-PROTOCOL$_{(wm)}$'s **sanitization** phase differs from its META-PROTOCOL$_{(hc)}$'s counterpart in that implementations of safety properties are executed. Nevertheless, the definitions of safety properties either consist of checks (i.e., `Mutual Exclusion`, `Membership`, `Origin`, `Identifier Unicity`, `Safe Rules`, `Execution Sequence Integrity`) or only modify the organization of the sanitized release but not its content (i.e., `Unlinkability`). The correctness of META-PROTOCOL$_{(hc)}$ is consequently preserved. As a result, all phases are correct and terminate, so META-PROTOCOL$_{(wm)}$ is totally correct. □

Theorem 10. Let $\pi_{(wm)}$ be the instantiation of the privacy algorithm $\mathbb{A}$ under META-PROTOCOL$_{(wm)}$. Given that $\pi_{(wm)}$ respects the security requirements, then $\pi_{(wm)}$ securely computes $\mathbb{A}$ against a weakly-malicious publisher.

Proof. A computationally-bounded weakly-malicious attacker cannot launch any active attack because he would have to use weakly-malicious actions as building blocks. As such actions will be detected by tokens, and because he is reluctant to being convicted as an adversary, he is reduced to launching passive attacks only. The security of $\pi_{(hc)}$ against passive attacks is stated in Theorem 7, and the safety properties's implementations that $\pi_{(wm)}$ additionally enforces are required *by definition* to preserve the protocol's security. As a result, active attacks are precluded and passive attacks are inoperative: $\pi_{(wm)}$ securely computes the privacy algorithm $\mathbb{A}$. □

5.4 Implementation Techniques for Satisfying Safety Properties

This section sketches possible implementation techniques for satisfying the safety properties. We designed them to be simple such that they demonstrate the feasibility of the approach; more sophisticated implementations are let to future work. First, we present the techniques for satisfying the safety properties that preclude copy actions (`Identifier Unicity`, `Membership`, and `Mutual Exclusion`). Second, we describe the way we enforce `Invariance`. Third, we explain how tokens can perform counts protected against tampering actions from the attacker. Fourth, the algorithm satisfying `Unlinkability` is detailed. Finally, we sketch a straightforward way to guarantee the `Execution Sequence Integrity`.

5.4.1 Identifier Unicity, Membership, Mutual Exclusion

Together, the `Identifier Unicity`, `Membership`, and `Mutual Exclusion` preclude the malicious copy actions.

The `Identifier Unicity` property states that each tuple collected must be assigned a unique identifier. This identifier can be any arbitrary bitstring provided that it is unique (at least with high probability), and that disclosing it to the publisher does not unveil the identity of the individual related to the tuple. For example, it can be the semantic encryption of an identifier assigned to the token, similarly to the implementation of its counterpart within PROTOGEN$_{(wm)}$, or a pseudo-random number with a very low collision probability (e.g., a MAC of this identifier). `Identifier Unicity` is asserted by checking that within a received partition each identifier is unique. If the identifiers are pseudo-random numbers, the insignificant probability of collisions and their limited impact (the publisher merely keeps a single tuple among the colliding ones) makes collisions negligible. The integrity of tuple identifiers is guaranteed by the `Origin` safety property so that the publisher cannot tamper them without being detected.

The enforcement of the `Membership` and `Mutual Exclusion` safety properties on a given dataset is based on letting the publisher sort the dataset on tuple identifiers before partitioning it (the sorting order, whether ascending or descending, is meaningless). Hence, the set of tuple identifiers that a partition is supposed to contain is merely defined by the *range* of tuple identifiers that the partition covers, i.e., the min and max identifiers of the partition's tuples. Observe now that (1) if the complete set of ranges is unique, and (2) if it does not contain any overlapping range, then `Mutual Exclusion` is guaranteed. As a result, we come to the following execution sequence,

that enforces `Mutual Exclusion` and `Membership`: first, the set of ranges is made invariant, then, a token checks that no range overlaps, and finally, any participating token downloads a partition (e.g., for sanitizing it), check that its range is equal to the corresponding invariant range, and assert that all the tuple identifiers are covered by the range. Checking the consistency between the invariant ranges and the received partition forces the publisher to produce and make invariant the set of ranges obtained from the *actual* underlying partitions.

5.4.2 Invariance

The `Invariance` property ties a label designating a data structure to the actual bitstring value of the data structure.

Absolute Majority

The simple implementation of `Invariance` that we propose is based on checking that each couple of (label, bitstring) to be made invariant has been sent *to the absolute majority of a designated subset of tokens.*

How would a publisher willing to break `Invariance` act? He would aim at making invariant two couples with identical labels but different bitstrings; so would have to send both to $(50\%+1)$ of the designated tokens: at least one token would consequently receive the two disagreeing couples, thereby detecting the dishonest action.

We require for simplicity that the population of designated tokens be statically defined before the protocol starts: each designated token must know that it has been designated, and all tokens must know the number of designated tokens (see Chapter 6 for practical ways of setting pre-required parameters). The designated population is chosen arbitrarily, with the following tradeoffs in mind: (1) the more frequently its members connect, the smaller the latency, and (2) the more numerous it is, the more robust to the failure of its members it is though the higher the latency. For example, in a medical application where patients are equipped with tokens, the designated population could be the tokens of the most frequently visited patients (e.g., elderly people), or the health professionals's tokens if they are also equipped with tokens.

Compared to the PROTOGEN's implementation of `Invariance`, this implementation has a latency cost constant (depending only on the behaviour of the designated population), and provides stronger security guarantees (not probabilistic).

`Invariance` of Large Data Structures

When making invariant a large data structure such as the partitioned dataset, the bitstring sent is a *digest* of the structure: a set containing for each partition the result of hashing its tuples. The digest provides a concise representation, and its consistency with the digested data structure is easily asserted: a token having downloaded a partition (e.g., for sanitization) merely hashes the partition's tuples and compares the result with the hash announced in the digest.

Recall from Section 5.4.1, that the set of partitions's ranges must also be made invariant. For convenience, we gather the digest and the set of ranges in the same data structure called the *summary*.

Counting for `Invariance`

The proposed implementation of `Invariance` requires *counting* the number of tokens having received the summary. The details concerning the secure count scheme appear in the next section (Section 5.4.3); we only assume for the moment that such scheme exists. This assumption done, counting for `Invariance` appears trivial: any designated token that connects downloads the summary and increments the secure counter associated to the summary received (if not done previously). Checking the `Invariance` of a given summary amounts to downloading the summary's counter, and checking that it is indeed associated to the given summary and has reached the minimum value (i.e., $50\% + 1$ of the designated population).

5.4.3 Secure Counts

A naive count scheme could consist in using locks to synchronize tokens on a shared counter: each participating token would lock the counter first, increment it then, and sign it finally. However, due to the possibly high number of tokens concurrently connected, this approach would suffer from prohibitive blocking delays. We rather propose a count scheme free from locks, inspired from traditional parallel tree-based sum schemes. It consists in gathering on the publisher unitary *count increments* sent by tokens, and in summing them up while ensuring the absence of tampering from the publisher.

Count Increments

The *count increment* is the unitary count data structure; it is primarily made of a count observed by a token. For example, the number of tuples from $\mathcal{R}$ appearing in

a downloaded partition forms the observed count encapsulated in a count increment data structure. For ease of discussion, we assume for the moment that the publisher does not tamper the count increments, and we concentrate on the way we sum them up. We will come back to the question of guaranteeing the integrity of the final sum.

When all the count increments have been gathered on the publisher (e.g., the count increments of the $\mathcal{R}$ tuples have been received for all the partitions), the publisher asks tokens to sum them up. The count increments gathered first are called the *initial count increments*. Assuming that a token has sufficient resources leads to the following simple sum scheme: the token downloads the complete set of initial count increments, sums them up, and uploads the sum. If the number of count increments overwhelms the resources of a token, then other sum schemes must be imagined. We describe below the *hierarchical sum scheme*, a basic tree-based sum scheme allowing to tailor the number of increments handled by each connecting token in a simple way.

Hierarchical Sum Scheme

The hierarchical sum scheme is a recursive process whose execution can be represented as a hierarchy made of n_l levels (as depicted in Figure 5.4). At each level of the hierarchy, a *partitioning step* and a *summing step* are executed, the output of the level i being the input of the level $i+1$.

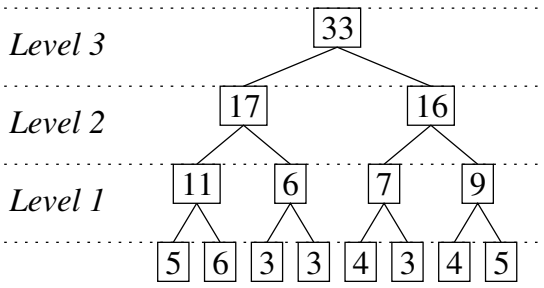


Figure 5.4: Example of a Hierarchical Sum (Unprotected against Tampering Actions)

The Sum Scheme. Let describe inductively the hierarchical sum scheme. During the level 1 of the hierarchy, the partitioning and summing steps are executed on the set of initial count increments gathered previously. First, the publisher performs the partitioning step: it partitions the count increments into *count partitions* such that each count partition fits the resources of a token. Second, the summing step is performed by tokens: each connecting token receives a count partition, sums up the count increments contained in the partition, and sends the sum to the publisher encapsulated in a new count increment. When all the partial sums of level 1 have been gathered (in the form

of count increments), level 2 is performed by executing the partitioning and summing steps on them. These two steps go on recursively, taking as input the count increments of level i and producing the count increments for level $i+1$. The process stops when a level produces a single count increment, which is the complete sum. Figure 5.4 depicts a hierarchical sum scheme in charge of summing up eight count increments, with each partition containing two increments.

Cost Analysis. The number of tokens connections required to perform the sum is the number of levels times the number of tokens connections required per level. Let first determine the number of level. Let n_c be the total number of count increments to sum up, and $n_{c/p}$ be the number of count increments per partition (fixed). Then the number of levels necessary to obtain the complete sum is $n_l = \lceil \log_{n_{c/p}}(n_c) \rceil$. Let determine now the number of tokens required for performing level i *at worst*. The number of tokens connections required for performing the level i being actually the number of partitions in level i, the worst case occurs when the number of partitions in level i is maximal; in other words, when the number of count increments at level i is a multiple of the partition's size. Level 1 requires: $\frac{n_c}{n_{c/p}}$ tokens connections, and level i with $1 < i \leq n_l$: $n_{c/p}$ times less than level $i-1$. Let lat_w denote the total number of tokens connections required at worst case, then $lat_w = \sum_{i=1}^{i \leq n_l} \frac{n_c}{(n_{c/p})^i}$.

Integrity of the Sum

The `Safe Rules` and `Invariance` safety properties rely on the sum. Guaranteeing its integrity is thus crucial. The publisher can produce a tampered sum by (1) forging count increments (increase the sum artificially), (2) copying count increments (increase the sum artificially), and (3) partially counting (decrease the sum artificially). Indeed, tampering actions are similar whatever the data structure, be it a set of tuples or a set of count increments. Note that the publisher does not perform a tampering action if it does not benefit from it; notably, it has no benefit in counting partially the increments for `Invariance`.

Forge Actions. The forge action is precluded by applying the `Origin` safety property to the count increments: each count increment is accompanied by its signature, which is then checked by tokens when they perform a sum.

Copy Actions: Purposes. Secure counts are a building block for the enforcement of the `Safe Rules` and `Invariance` safety properties. Obviously, count increments gathered, e.g., for `Safe Rules` must not be (maliciously) copied into the set of count increments gathered, e.g., for `Invariance`. Tokens consequently assign a *purpose* to each count increment that they form. A purpose is basically a bitstring. The set of

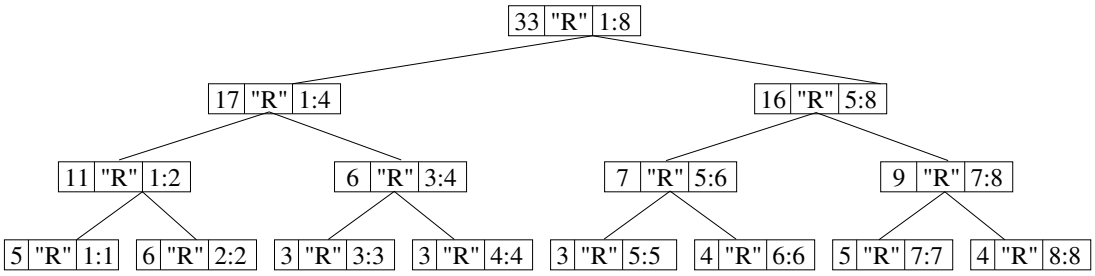


Figure 5.5: Example of Hierarchical Sum (Protected against Copy Actions)

purposes is finite and known by all tokens as well as the association between each purpose and the corresponding execution step. When receiving a count partition, tokens check that all the count increments embed the same purpose, and when returning the sum, the encapsulating count increment embeds the same purpose as the input count increments. Consequently, count increments gathered for enforcing different purposes cannot be summed up together anymore. Note that the purpose additionally includes a hash of the data structure being counted when the latter is not invariant (typically, a hash of the summary *in the process* of being made invariant).

Copy Actions: Intra/Inter-Partition Copies. Second, copy actions can also occur within a set of count increments gathered for the same purpose. These copy actions are either intra-partition or inter-partition copies; they are similar to their counterpart affecting the dataset, so precluding them involves similar measures:

- Each initial count increment is assigned a unique identifier expressed as a single-value range (the nature of identifiers is detailed below);
- The initial count increments are sorted according to their identifier before being input by the level 0 of the sum scheme;
- A token receiving a count partition checks that, among the set of count increments, no range overlaps, and sums the counts observed. The identifier of the sum's count increment is the range covering the identifiers appearing in the partition.

As a result, copy actions are detected as soon as the two partial sums that are assigned overlapping ranges appear within the same partition. Figure 5.5 depicts the sum hierarchy of Figure 5.4 protected against copy actions. The leftmost cell of each count increment contains its count observed, the middle cell contains its purpose (in the example, the purpose is set to "$\mathcal{R}$", i.e., counting the $\mathcal{R}$ tuples), and the rightmost cell contains the range of identifiers covered.

What are the identifiers of the initial count increments? Their nature actually depends on what is counted. Indeed, assume that tokens assign random identifiers to

the count increments of `Safe Rules`. By sending twice the same partition to tokens, the publisher would obtain twice the same count (i.e., a copy) but to which are assigned distinct identifiers with high probability: the sum would be tampered. Hence, for `Safe Rules`, the count increment resulting from counting the *same* partition must embed the *same* identifier: the count increments's identifiers are merely the partitions's TID-Sets. Nevertheless, random identifiers perfectly fit the `Invariance`'s count increments. Indeed, a token having already participated in `Invariance` will not return a count increment again: the publisher cannot obtain twice the same count.

Counting Partially. We have seen above how to preclude the publisher to increase the sum artificially. Let us now focus on precluding it to *decrease* the sum artificially. The attacking publisher may benefit from summing a *partial* set of initial count increments for thwarting the `Safe Rules` property. For example, decreasing the sum related to $\mathcal{D}_o$ tuples results in jeopardizing the privacy guarantees of the $\alpha\beta$-Algorithm. Guaranteeing the completeness of this sum requires checking that the count increments resulting from *all* the partitions have been summed up. This is easily achieved by embedding within each count increment the number of partitions that it covers. For initial count increments, this number is set to 1; for count increments resulting from a partial sum, it is set to the number of initial partitions involved in the sum. When checking the `Safe rules` property, tokens check that the sum covers the exact number of partitions (tokens checking `Safe Rules` know the actual number of partitions from the invariant summary).

Resulting Count Increment. Count increments consequently adopt the following form: $c \leftarrow (o, p, r, n?)$, where $c.o$ denotes the count observed (an integer), $c.p$ denotes the purpose (a bitstring), $c.r$ denotes the range of count increments covered (a range of integers), and $c.n$ (optional) denotes the number of count increments covered (an integer).

Security Established. The security of the count scheme is established, as previously, by showing that basic malicious actions are precluded, thereby precluding sophisticated attacks (which are arbitrary combinations of basic actions). First, let us consider the sum dedicated to the `Invariance` safety property. The attacker has no benefit in decreasing artificially the sum; he aims, on the contrary, at increasing it. To this end, he must either forge count increments, or copy them. However, both actions are precluded: count increments are signed (no forge actions), and embed identifiers (no intra/inter-partition copies of count increment). The sum for `Invariance` can thus be securely computed. Let us now consider the sum dedicated to the `Safe Rules` safety property. In addition to increasing it artificially, the attacking publisher could benefit from decreasing it. Artificial increases are precluded in a similar way to previously, while artificial decreases are precluded by forcing the publisher to sum

the count increments of *all* partitions. The sum for `Safe Rules` can thus be securely computed.

5.4.4 `Unlinkability`

Informally speaking, the `Unlinkability` safety property requires to disable the mapping between (a subset of) sanitized records and (any subset of) clear and auxiliary parts. Intuitively, our proof-of-concept implementation of `Unlinkability` consists in the following: sanitized tuples are uploaded on the publisher in an encrypted form, shuffled by tokens, and finally decrypted to form $\mathcal{V}$. As a result, no sanitized record in $\mathcal{V}$ can be mapped to a subset of clear and auxiliary parts. The intermediate dataset holding the encrypted sanitized tuples is called the *obfuscated sanitized release* $\mathcal{V}_o$.

Shuffling with Tokens

How can lightweight tokens shuffle the complete obfuscated sanitized release? The proposed shuffling protocol copes with this (1) by having each connecting token shuffle a constant number of tuples, and (2) by progressing incrementally such that the output of a higher incremental step, or *level*, be equivalent to shuffling a larger number of tuples.

All levels are similar in spirit: each connecting token downloads, say f, sets of encrypted tuples - called *buckets* - resulting from the preceding level, decrypts, shuffles, and re-encrypts the tuples, and outputs f buckets (all buckets have the same, fixed, cardinality). Hence, because tuples have been both re-encrypted (so that, at each re-encryption, they are indistinguishable from a new random bitstring) and shuffled, any tuple in the buckets output can correspond to any tuple in the buckets input. The parameter f is called the *branching factor*. Together, the total size of the f buckets is equal to the size of a partition.

Intuitive Description. The *shuffling circuit* determines the organization of each level by choosing the buckets that should be shuffled together by each connecting token. To this end, at each level, each bucket is assigned a position, where at level 0, the position of a bucket is simply its position in the obfuscated release ordered by tuple identifiers (positions start at 0). Figure 5.6 depicts the shuffling circuit of four partitions (i.e., partitions $\mathcal{P}_1$ to $\mathcal{P}_4$), each made of four tuples, with $f = 2$ (`E` denotes the semantically secure encryption operation, implicitly parametrized by the usual cryptographic material). The buckets's positions are preceded with a #. The shuffling circuit determines for each level the buckets shuffled together based on their positions: level i consists in shuffling together the buckets (1) output by level $i-1$ if $i > 0$, and (2) whose positions differ by f^i (starting at position #0).

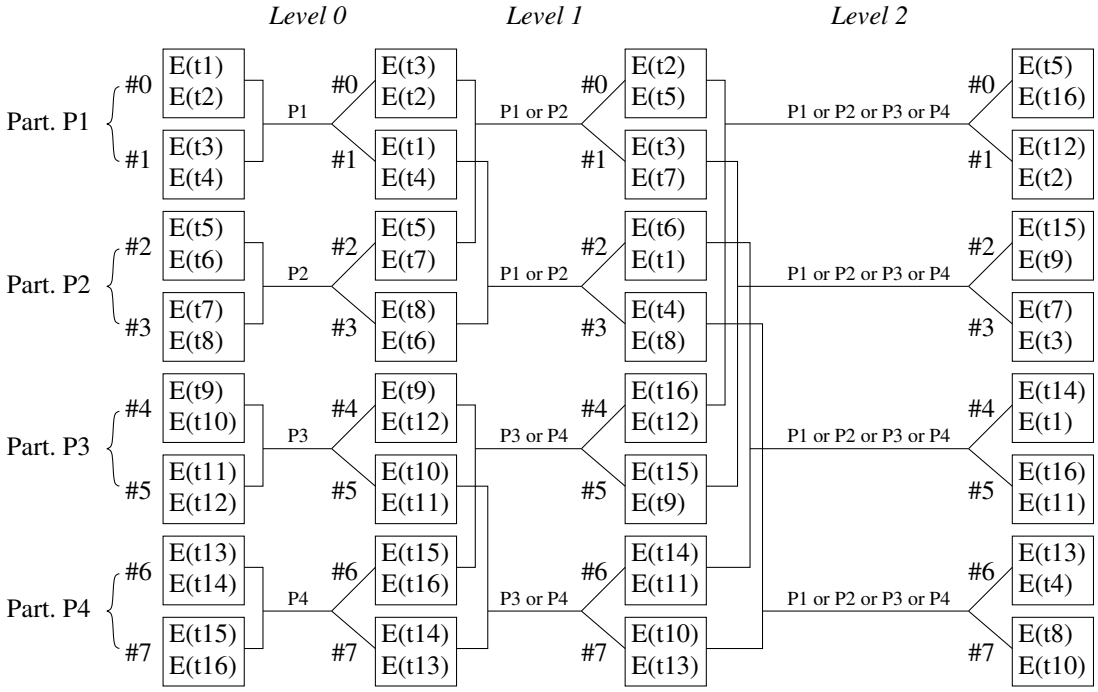


Figure 5.6: Example of a Shuffling Circuit

For example on Figure 5.6, the level 0 buckets #0 and #1 are downloaded by a connecting token, which decrypts their tuples, shuffles, re-encrypts, and returns them to the publisher. Any tuple returned by this token is thus likely to correspond to any tuple of the initial partition $\mathcal{P}_1$ (which consists in the buckets #0 and #1 of level 0). Similarly, the level 1 buckets #0 and #2 are shuffled together, resulting in tuples that correspond to any tuple of the initial partitions $\mathcal{P}_1$ or $\mathcal{P}_2$. Finally, the level 2 outputs tuples that correspond to any tuple of ($\mathcal{P}_1$ or $\mathcal{P}_2$ or $\mathcal{P}_3$ or $\mathcal{P}_4$).

Indeed, it appears that level i outputs buckets into which tuples can correspond to the initial tuples from f^i distinct partitions. The number of partitions to which each tuple output by level i can correspond is called the *scope* of level i. When the scope is as high as the initial number of partitions, the shuffling is complete.

For example on Figure 5.6, at the end of the second level the scope is equal to $2^2 = 4$, which is the number of initial partitions; the shuffling is thus complete.

Input Bucket Positions. Bucket positions within levels are crucial to the shuffling circuit. Let us start by defining, for each level i, the input buckets that must be shuffled together (e.g., on Figure 5.6 at level 1, the input buckets {#0, #2} must be shuffled together, as well as the input buckets {#1, #3}, {#4, #6}, and {#5, #7}). As said above intuitively, at level i, the input buckets shuffled together are determined

based on their positions.

In order to favor a clear understanding, we start by focusing on the simple case where $f = 2$, and we generalize after for an arbitrary f value. Let i denote the current level and *pos* denote the position of the given bucket. Then the position of the bucket with which it must be shuffled is:

$$\lfloor pos/2^{i+1} \rfloor \cdot 2^{i+1} + ((pos + 2^i) \mod 2^{i+1}) \tag{5.1}$$

Let us explain intuitively the above formula. It is made of an absolute position (the left member) and an offset relative to the absolute position (the right member). We can see the input of any level i as divided into sets of 2^{i+1} buckets (i.e., the scope - intuitively defined above). The current bucket belongs to one of these sets - it must be shuffled to another bucket that belongs to the same set. The left member determines the beginning position of the given bucket's set (i.e., $\lfloor pos/2^{i+1} \rfloor \cdot 2^{i+1}$). The right member determines the *round-robin* offset of the bucket to be shuffled, inside the given bucket's set (observe the (modulo) shift of 2^i positions).

For example, on Figure 5.6 level 1, assume that we want to determine the bucket that has to be shuffled with bucket #6. Its position is given by the formula: $\lfloor 6/2^{1+1} \rfloor \cdot 2^{1+1} + ((6 + 2^1) \mod 2^{1+1}) = 4 + 0 = 4$. This equation says that the bucket to shuffle is in the set that starts at position #4 (i.e., the left member), at an offset 0 (the right member).

More generally, whatever the position of a given input bucket, the positions of the buckets with which it must be shuffled are defined by the following formula:

$$\bigcup_{1 \leq j \leq f-1} \lfloor pos/f^{i+1} \rfloor \cdot f^{i+1} + ((pos + j \cdot f^i) \mod f^{i+1}) \tag{5.2}$$

Output Bucket Positions. As said above, the buckets's positions at level 0 are easily determined by the initial partitions's order. However, when considering higher levels, fixing the position of a shuffled bucket is less trivial. The positions of shuffled output buckets depend on the positions of the buckets that were input, on the current level, and on the branching factor. Let pos_{min} denote the minimal position among the buckets input, *pos* the position of an arbitrary bucket input (which one does not matter), and i the current level. The positions of output buckets are then comprised within a range starting at the position pos_{start}, where pos_{start} is determined by

$$\lfloor pos/f^{i+1} \rfloor \cdot f^{i+1} + (pos_{min} \mod f^i) \cdot f \tag{5.3}$$

and ending $f - 1$ positions later (included): $pos_{end} \leftarrow pos_{start} + (f - 1)$. The precise position of an output bucket within the range $[pos_{start}, pos_{end}]$ does not matter because the distribution of tuples into buckets is random.

Let us explain briefly the pos_{start} formula. Similarly to the formula in charge of defining the input buckets, it is made of an absolute position (still the left member) and an offset relative to the absolute position (still the right member), and we still consider the input of any level i as divided into sets of f^{i+1} buckets. Given input buckets to be shuffled, the left member of the pos_{start} formula (i.e., $\lfloor pos/f^{i+1} \rfloor \cdot f^{i+1}$) determines the starting output position of the set of f^{i+1} buckets to which belong the input buckets (this is the absolute position). The right member (i.e., $(pos_{min} \mod f^i) \cdot f$) determines where the f buckets output should be positioned within this set (this is the relative offset). For example, on Figure 5.6 level 1, let consider the shuffling of the {#1, #3} buckets. Here, the absolute position is 0 (the output buckets are in the 0^{th} set of 2^2 buckets), and the relative offset is 2 (the position of the output buckets should start at the third position of this set): the resulting position is $pos \in [2, 3]$.

Arbitrary Number of Partitions

The above scheme assumes that the number of partitions in the dataset to shuffle is a power of the branching factor. Indeed, if this is not the case, some tuples will remain on the down-side of the shuffling circuit, without having corresponding tuples to be shuffled with. Figure 5.7 depicts the case where the dataset is made of five partitions and $f = 2$. The level 1 #8 and #9 buckets have no corresponding buckets to be shuffled with. Indeed, shuffling them with buckets from the level 0, then from the level 1, and finally from level 2 would yield duplicate tuples and consequently thwart the probabilities; and shuffling them directly with a bucket from level 2 would not yield duplicates but probabilities would still be thwarted. For example, assume that the level 1 #8 bucket is shuffled with the level 2 #7 bucket. The resulting tuples have a probability 1/2 to correspond to the tuples of the initial partition $\mathcal{P}_5$ and a probability 1/2 to correspond to the other partitions: correspondence probabilities between initial and output tuples are not uniform.

A naive way to cope with an arbitrary number of partitions is to let the publisher insert clearly identified *artificial* tuples into the dataset before starting the shuffling circuit such that the number of partitions becomes a power of f. Artificial tuples are then removed after the shuffling, tokens ignoring them when disclosing the final sanitized release. However, this may result in a high overhead because of the possibly high number of tuples to insert. Let $|\mathcal{V}_o|_p$ denote the number of partitions in the obfuscated sanitized release $\mathcal{V}_o$. For example, if $|\mathcal{V}_o|_p = f^{10} + 1$, the publisher would have to insert $f^{11} - (|\mathcal{V}_o|_p + 1)$ false partitions in order to reach the next power of f. This would result in a dataset much bigger than the initial one, so requiring much more tokens's connections to be shuffled.

How can we reduce the number of artificial tuples to insert? We answer this question

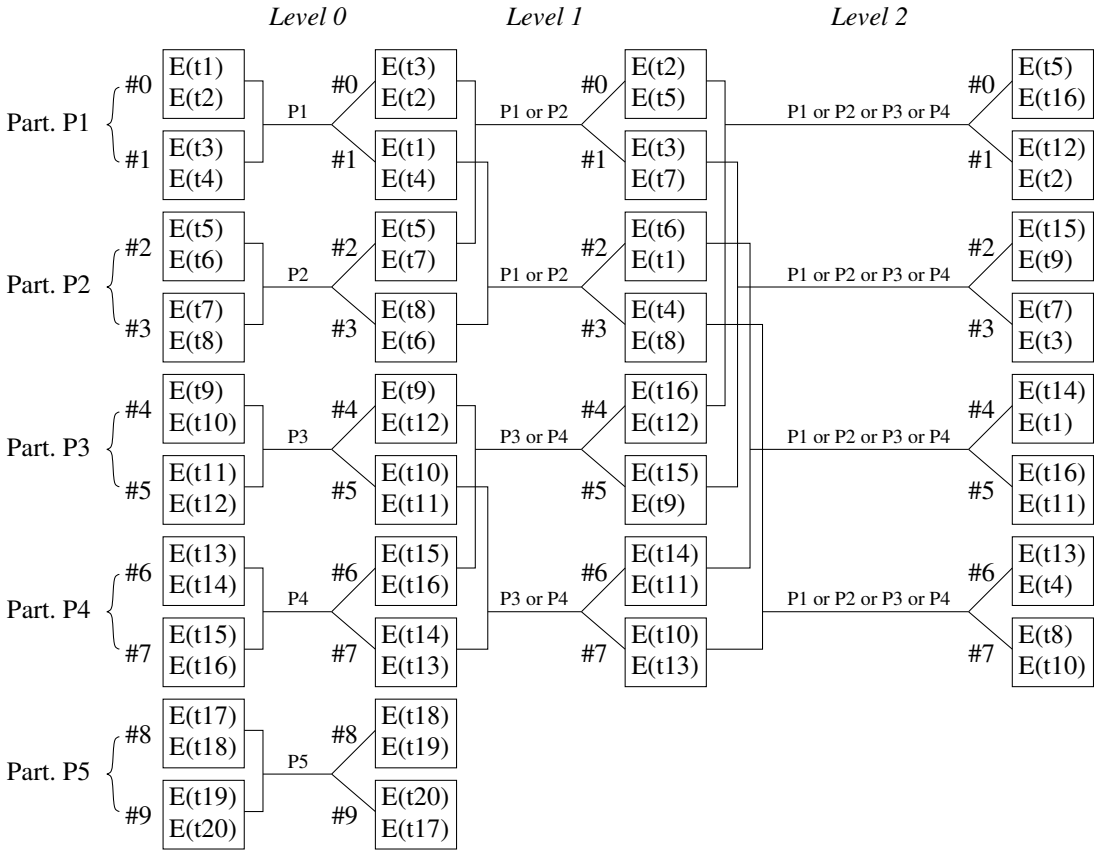


Figure 5.7: Incomplete Shuffling Circuit

by allowing the branching factor to decrease during the last shuffling level. Our solution requires essentially that (1) the branching factor be greater than 2 so that it can be decreased during the last level, and (2) the number of partitions to shuffle be a *multiple of a power of* f (and not necessarily a power of f anymore). The requirement (1) does not place any constraint on the dataset in practice because it is easily satisfied , while the requirement (2) results in the insertion of a reduced number of false partitions.

Let n_l denote the total number of levels. The shuffling circuit is now as follows. All levels except the last are the same as previously (i.e., they output tuples corresponding to $f^{(n_l-1)}$ initial partitions) because the number of partitions is a multiple of $f^{(n_l-1)}$: $|\mathcal{V}_o|_p = m \cdot f^{(n_l-1)}$, where $1 < m \leq f$. The last level consists in setting the branch factor to m, and shuffling m by m the buckets output by level $(n_l - 1)$.

Figure 5.8 illustrates the case where four partitions must be shuffled together and $f = 3$. The requirement (1) is fulfilled, and the requirement (2) forces the insertion of two partitions (in black in the figure) such that $n_{\mathcal{V}_o/p} = m \cdot 3^1$ with $m = 2$. During level

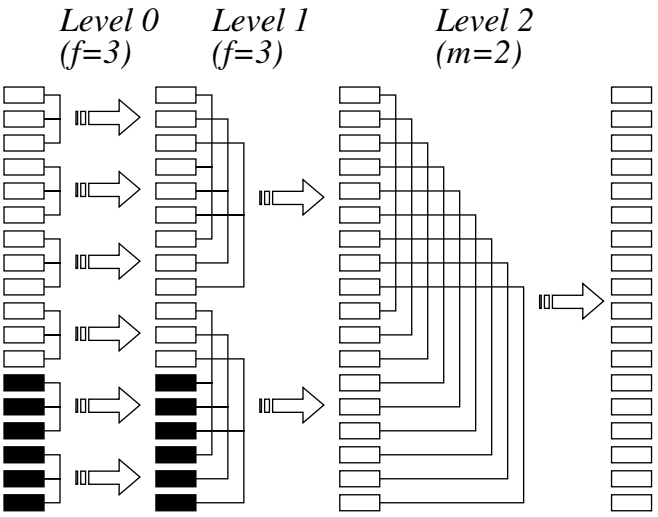


Figure 5.8: Complete Shuffling Circuit of an Arbitrary Number of Partitions

1, the partitions are shuffled f by f ($f = 3$); during the last level, they are shuffled m by m ($m = 2$).

The formula used for computing the positions of output buckets would need some adjustments to be valid with a decreased branch factor. However, the positions of buckets output by the last level are trifling since the shuffling circuit is over: up to the publisher to organize them as it wishes.

Cost Analysis

The cost is the total number of tokens's connections required; we denote it lat. The latency of each level is the number of buckets ($n_b \leftarrow f \cdot |\mathcal{V}_o|_p$ remains constant along the shuffling circuit) divided by the branching factor (f for all levels except the last, for which it is equal to m). The last level is apart because it is executed based on a possibly decreased branching factor. Let n_l be the number of shuffling levels. It is appearant that $n_l = \log_f(n_b) = \log_f(|\mathcal{V}_o|_p) + 1$. Consequently, and after factorization, the total number of connections required is: $lat = |\mathcal{V}_o|_p \cdot (\log_f(|\mathcal{V}_o|_p) + f/m)$. The shuffling protocol thus exhibits a linearithmic cost $O(|\mathcal{V}_o|_p \cdot \log_f(|\mathcal{V}_o|_p))$.

Correctness and Security

Correctness. The correctness of the shuffling protocol is established by showing that within a given level (except the last), (1) each bucket is shuffled with the f other buckets whose position differs from a multiple of f^i positions (round-robin in the same

scope), (2) each bucket is not shuffled with any other bucket whose position differs from at most $f^i - 1$ positions, and (3) all buckets outputs are assigned a distinct position based on which the following level will shuffle them with buckets from another scope. The correctness of the last level is established similarly concerning the positions of its input and does not need to be established concerning the position of its output. We show below that the three points above are checked: the shuffling protocol is correct.

Let us consider Point (1). We start by observing in Equation 5.2 that $f-1$ buckets will be shuffled with the given bucket (see the inequation below the $\cup$ symbol). Next, we observe that the positions of all buckets of the same scope have the same left member (i.e., $\lfloor pos/f^{i+1} \rfloor \cdot f^{i+1}$), but a varying right member (i.e., $(pos + j \cdot f^i) \mod f^{i+1}$). The latter starts at a position which is f^i greater than the given bucket's position, and increases by steps of f^i as j increases. From these two observations, we conclude that the given bucket is shuffled with f buckets, whose positions differ from a multiple of f^i positions (round-robin in the same scope). Since, the multiple j is lower than f, and the modulo is f^{i+1}, then $j \cdot f^i$ never performs more than one round modulo f^{i+1}. As a result, the f buckets shuffled together are all distinct.

Let us consider Point (2). We start by observing in Equation 5.2 that two buckets that are not in the same scope will not be shuffled together. Indeed, the left member yields the beginning position of the scope, and the right member does not yield offsets greater than f^{i+1}. Then we see that, within the same scope, two positions differing from at most $f^i - 1$ are assigned two disjoint sets of positions to be shuffled with. Indeed, observe that in Equation 5.2, though the left member is the same for both input positions, their respective right member will never be the same, whatever j, because both positions differ from at most $f^i - 1$ positions and progress by steps of f^i modulo f^{i+1}. As a result, Point 2 is verified.

Finally, let us consider Point 3. As previously, we first observe in Equation 5.3 that positions of buckets within the same scope have the same left member (still the beginning position of the scope) but have varying right members. Given an input set of buckets to be shuffled together, the right member is determined by the offset of the minimal position in the input with respect to the scope of the minimal positions. Since, each minimal position appears in a single set to shuffle, and each shuffled output is booked f output positions (see the multiplication of the offset by f in the right member), then output buckets within the same scope are assigned distinct positions. We then observe that two input sets belonging to distinct scopes cannot be assigned the same position: their left members are the beginning positions of their respective scopes, and their right members is lower than f^{i+1}. Finally, we observe that output positions are within the same scope of their input positions. Because the following level increases the scope of the shuffling, it will shuffle input buckets from two different

scopes, that have not been shuffled together yet. As a result, Point 3 is checked.

Security. The shuffling protocol is secure because (1) the integrity of its input, i.e., the obfuscated sanitized release, is protected (see Section 5.4.5), and (2) the integrity of its execution sequence is protected by the `Execution Sequence Integrity` (see Section 5.4.5).

5.4.5 `Execution Sequence Integrity`

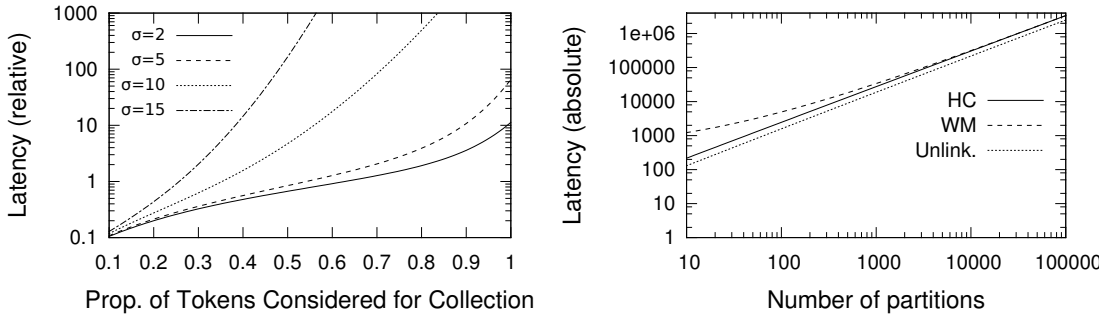
The `Execution Sequence Integrity` safety property prevents the publisher from tampering the execution sequence primarily based on *proofs* (in its broadest meaning). We informally describe below how we implement such proofs (formal statements would only make the presentation cumbersome without clarifying it).

We propose to instantiate these proofs based on usual private-key signatures; emitted by a token having performed a given step on a given data structure, a signature binds the step's *label* (e.g., a string of characters) to the data structure (possibly through, e.g., a hash). For example, when a token checks the `Mutual Exclusion` property over a given summary $\mathcal{S}$, it returns to the publisher: $\mathtt{M}_\kappa$("Mutual Exclusion" || $\mathtt{H}(\mathcal{S})$), where || denotes the bitwise concatenation. When a token is asked to treat a partition from $\mathcal{P}$, it downloads $\mathcal{P}$'s summary $\mathcal{S}$, the `Mutual Exclusion` proof p_{mutex}, and checks the validity of the proof: $p_{mutex} = \mathtt{M}_\kappa($ "Mutual Exclusion" || $\mathtt{H}(\mathcal{S}))$. Similar checks occur all along the execution sequence (including the shuffling circuit), which is thus completely chained by signatures. We do not detail further this implementation which is rather trivial.

A specific `Execution Sequence Integrity` check occurs for chaining $\mathcal{P}$ and $\mathcal{V}$ through the obfuscated sanitized release $\mathcal{V}_o$ and the shuffling circuit. To guarantee the protection's continuity, the obfuscated sanitized release must enforce the usual safety properties against tampering actions (no copy/forge/delete), and must also be *complete*: i.e., the number of (non-artificial) tuples contained in $\mathcal{P}$ and $\mathcal{V}_o$ must be equal. Algorithms that modify the dataset's cardinality make a straightforward use of the `Count` mechanism to allow checking the completeness of $\mathcal{V}_o$ (i.e., for the $\alpha\beta$-Algorithm, `Count` the tuples from $\mathcal{D}_o$ having been dropped during $\mathcal{P}$'s sanitization, those remaining in $\mathcal{V}_o$, and compare them to the total number of tuples of $\mathcal{D}_o$ that appeared in $\mathcal{P}$).

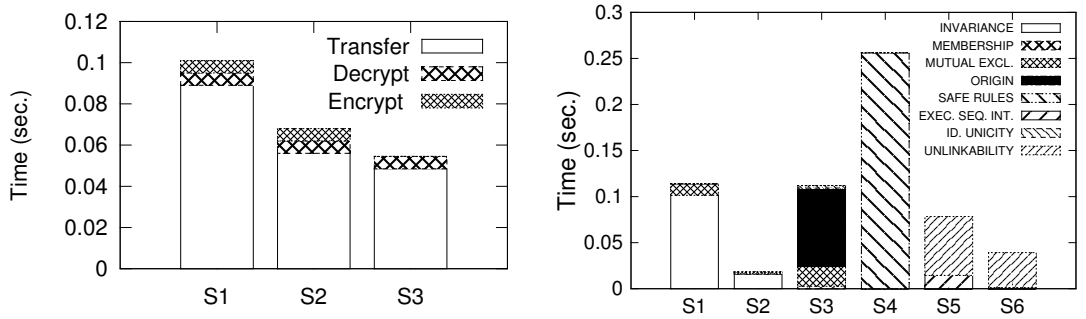
5.5 Experimental Validation

This section validates experimentally the feasibility of our approach by studying the Meta-Protocol's costs in terms of *latency* (i.e., the number of token connections



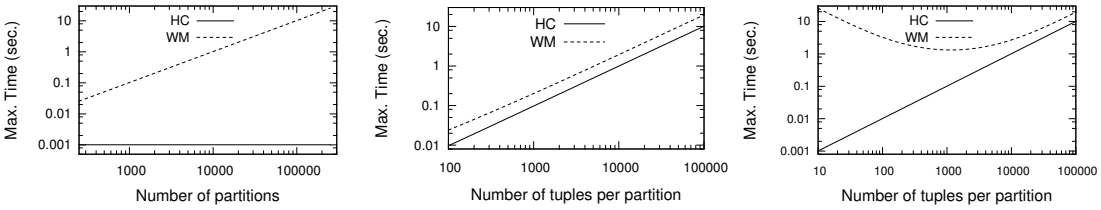
(a) Latency (Collection)

(b) Latency (Sanitization)



(c) Times (HC, 1K Tup./Part.)

(d) Times (WM, 1K Tup./Part., 2.5KPart.)



(e) Scaling (10 Tup./Part.)

(f) Scaling (10 Part.)

(g) Scaling (Total 1M Tup.)

Figure 5.9: Latency and Internal Time Consumption

required to perform each execution step) and *internal time consumption* (i.e., the time required by a token to perform each unitary task). Although the experimental results are obtained through the $\alpha\beta$-ALGORITHM instantiation of the META-PROTOCOL, the tendencies observed are general and independent from any specific algorithm. We have implemented the META-PROTOCOL and are currently integrating it in the PlugDB prototype [Anciaux10] described in Chapter 6. The experimental environment set for validating the META-PROTOCOL is similar to the one set for PROTOGEN in the Chapter 4.

5.5.1 Latency

We measure the latency of the collection and sanitization phases in terms of number of tokens' connections required to perform them (the construction phase's latency is negligible compared to them). Further conversions of the latency into any time unit can be obtained by modeling the application-dependent time variations of tokens's connections. The collection phase's latency depends on *the percentage of distinct tokens* that are required to connect in order to form the initial dataset; measuring it obviously assumes a specific distribution of tokens' connection probabilities. The sanitization phase's latency is free from this assumption; it depends on *the number of partitions* in the partitioned dataset. As a result, we study the latencies of the collection and sanitization phases according to the parameter they respectively depend on.

Collection Phase

Settings. To make the presentation easier, we represent the population of tokens by a set of integers, each token corresponding to an integer between 0 and $(n_\Theta - 1)$, where n_Θ is the total number of tokens. We model the connection probabilities by a Gaussian distribution whose expectation μ is fixed to the halfway integer, i.e., $n_\Theta/2$, and whose standard deviation σ is a variable parameter. For each connection, a random sampling from this distribution determines the token connected. We simulate different applicative behaviors by varying the standard deviation: a random sample - i.e., a token that connects - has, e.g., 95% chance of lying in the range $[\mu - 2\sigma, \mu + 2\sigma]$, 68% chance in the range $[\mu - \sigma, \mu + \sigma]$, etc.

Results. Figure 5.9(a) shows the latency of the collection phase according to the percentage of distinct tokens required to connect in order to form the initial dataset. We express the latency as the proportion of (non distinct) token connections having occurred with respect to the total number of tokens. We plot the latencies corresponding to four distributions of token connections (i.e., $\sigma = n_\Theta/2$, $\sigma = n_\Theta/5$, $\sigma = n_\Theta/10$,

and $\sigma = n_\Theta/15$) with respect to the proportion of tokens considered by the **collection** phase (in other words the proportion of distinct tokens required to connect for forming the initial dataset). This figure highlights the exponential cost for reaching a predefined percentage of the population.

Sanitization Phase

Results. The latency of the sanitization phase is free from the assumption about the distribution of tokens' connections. It is expressed as the absolute number of (non distinct) required token's connections and varies according to the number of partitions to sanitize. Figure 5.9(b) shows the linear evolution of the sanitization phase's latency according to the number of partitions (independently from the partition's size). Observe the domination of the cost of the `Unlinkability`'s implementation: the gradual shuffling of the encrypted sanitized records results in its partitions being downloaded several times to be completely shuffled. The latencies of both the honest-but-curious and weakly-malicious protocols are similar because primarily made of the `Unlinkability`'s latency.

5.5.2 Internal Time Consumption

Settings (Dataset). The data schema is an extract of the CENSUS[23] dataset's schema often used in PPDP works; it consists of the attributes {*Age (78 distinct values), Education (17), Marital (6), Race (9), Work-Class (10), Country (83), Salary-Class (50)*}, where the domain of each attribute has been mapped to the domain of integers. Indeed, neither the Meta-Protocol nor the $\alpha\beta$-Algorithm is sensitive to the actual values of collected tuples; only *the number of partitions* and *the size of a partition* have an impact. The initial dataset contains up to 1 million records. The privacy parameters of the $\alpha\beta$-Algorithm were chosen following the authors [Rastogi07] (their precise values do not matter here) and result in the insertion of approximately $1.5 \cdot n_\mathcal{D}$ random records (i.e., up to 1.5 million random records inserted).

Results. Figure 5.9(c) depicts the maximum internal time consumption of the $\alpha\beta$-Algorithm$_{(hc)}$'s execution steps, grouped by basic operation, each partition containing 1K tuples (no matter 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 denote it the basic cost. The total cost remains under one second and is mainly made of the transfer cost, confirming the affordable costs of cryptographic operations. Figure 5.9(d) plots the maximum internal time consumption of

[23] http://ipums.org/

the $\alpha\beta$-ALGORITHM$_{(wm)}$'s main execution steps (as depicted in Figure 5.3), grouped by safety property, for 2.5K partitions in the partitioned dataset, each containing 1K tuples. We obtained the measures based on a single token: it thus downloaded $\mathcal{P}$'s summary only once, i.e., during the Step S1, and stored it in order to perform the other checks that require the summary (i.e., in Step S2 for checking that it was made invariant, and in Step S3 for asserting that `Mutual Exclusion` was checked on it). We observe that the safety properties overhead is primarily due to `Invariance` during Step S1, and `Safe Rules` during Step S4. Indeed, during Step S1 tokens download the digest data structure of the dataset to be made invariant (i.e., one hash value and one integer per partition), and also perform the sum of the count increments collected for `Invariance`. The cost exhibited by Step S4, high but still affordable, is due to the computation of the sum for `Safe Rules` based on a single count increments partition, resulting in a large data structure (i.e., two count increments per partition). If the cost of handling a single partition would have been prohibitive, the count increments partition would have been split into several partitions, and summed up by the hierarchical scheme described in Section 5.4. The other safety properties exhibit negligible costs.

Figure 5.9(e) shows how vary the maximum internal time consumptions of both versions of the protocol with respect to the total number of partitions (the size of a partition being fixed to a negligible quantity - 10 tuples): while the honest-but-curious version of the protocol remain unimpacted by these variations, the cost of the weakly-malicious safety properties varies linearly according to the linear increase in size of the data structures made invariant and of the number of counts to sum up for checking the rules. Figure 5.9(f) 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 weakly-malicious safety properties that depend on the partition's size. Finally, Figure 5.9(g) considers a fixed dataset of 1 million 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 weakly-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.

5.6 Synthesis

The META-PROTOCOL is the result of a deep re-thinking of the key findings of PROTOGEN freed from its simplifying assumptions.

Tackling the honest-but-curious publisher was the opportunity to settle the shape of the execution sequence. Tuples collected during the collection phase follow a generic

form where the information voluntarily disclosed to the publisher does no consist in quasi-identifiers anymore but is let unspecified (though constrained by security and functional requirements). The construction phase does not consist necessarily in forming equivalence classes; it is let unspecified too. Finally, the unitary data structure handled by tokens during the sanitization phase is not the equivalence class anymore but a self-contained arbitrary set of tuples called a partition. The inferences introduced by the partition-based processing model are disabled by the new `Unlinkability` safety property. We gave the conditions under which a given algorithm can be supported by the Meta-Protocol's execution sequence, and illustrated its genericity level by instantiating the key data structures under a wide and various range of privacy algorithms. Then, we tackled the weakly-malicious attack model by dissociating the Protogen's safety properties from the notion of equivalence class, analyzing the `Safe Rules` property to allow it to encompass (at least) any model based on counting some algorithm-dependent data, and defining the new `Execution Sequence Integrity` property in charge of guaranteeing that the publisher performs the execution sequence completely and in the order expected. We followed by describing proof-of-concept implementations of the safety properties, designed in favoring simplicity rather than efficiency. Finally, we validated experimentally the Meta-Protocol based on its $\alpha\beta$-Algorithm instantiation. We conclude that the Meta-Protocol suite achieves convincingly the **Decentralization**, **Genericity**, and **Generality** objectives of this thesis.

Chapter 6

Practical Adequacy

Summary: *In this chapter, we sketch a possible implementation of an* electronic health record *system based on tokens, in which we root a discussion of the practical aspects related to the setting and execution of the* Protogen *and* Meta-Protocol *suites of protocols. The architectures proposed by the current electronic health records projects worldwide range from completely centralized health records to token based approaches. Our approach tackles their limits, proposing a complete patient-centered architecture based on both tokens and supporting central servers. The contributions of this chapter are precisely this architecture, and the practical discussion about setting the parameters of the protocols proposed in this thesis and executing them. The content of this chapter includes (but is not limited to) [Allard09, Allard10c, Allard10b].*

Contents

6.1 Introduction

Chapters 4 and 5 have focused on the technical aspects of the Protogen and Meta-Protocol suites; the current chapter discusses the practical aspects related to a concrete implementation of these protocols. The use case around which we build the discussion is highly topical. Driven by the need to improve the quality of care while decreasing costs, many countries around the world are setting up large scale *electronic health record* systems (*EHR* for short) gathering the medical history of individuals. This chapter outlines a possible implementation of an EHR system based on tokens, in which it then roots a practical discussion about the privacy-preserving data publishing protocols proposed in this thesis.

Electronic health record systems are being launched worldwide. A recent report identified more than 100 EHR running projects at the scale of a country or a region in 2007 [Door08]. Other reports suggest that about 25% of US healthcare practices use EHR systems, and building a national health record system is currently a priority for the US administration [Mearian09]. Within Europe these figures vary greatly between countries, from 15% in Greece up to 90% in the Netherlands today. Indeed, EHR systems carry many promises ranging from an improved quality of care (e.g., patient information is accurate, up-to-date, and available) to large cost reductions (e.g., no redundant expensive tests).

However, despite their unquestionable benefits, studies conducted in different countries show that patients are reluctant to use existing EHR systems arguing increasing threats on individual privacy [Rose08, Bos09]. This suspicion is fueled by computer security surveys pointing out the vulnerability of database servers against external and internal attacks [Gordon06]. Indeed, centralizing and organizing the information makes it more valuable, thereby motivating attacks and facilitating abusive usages. Regardless of the legislation protecting the usage of medical data and of the security procedures put in place at the servers, the patient has the sense of losing control over her data.

This chapter suggests a new way of organizing an EHR system. The solution proposed capitalizes on the opportunities opened by tokens. Indeed, embedding a database system and a web server in a token allows to manage securely a healthcare folder in complete autonomy. Contrary to the rest of this thesis, this chapter focuses on a token implemented in a USB stick form factor. Accessing the on-board folder at patient's home thus requires a simple rendering device (e.g., a netbook or PDA) equipped with a USB port and running a web browser. The token's security guarantees (tamper-resistant hardware, certified embedded software) provide a much higher confidence than any traditional server can provide. However, a patient's folder stored

in her token cannot be accessed without being physically in possession of the token. We propose to reintroduce in the architecture *supporting servers* in charge of enabling secure exchanges of information (1) between the tokens of the patient and of a trusted circle of persons (e.g., the family doctor expressing her opinion in an emergency situation without having the patient's folder on hand), and (2) for the privacy-preserving data publishing protocols proposed in this thesis. Based on this architecture, we discuss practical ways to setting the various parameters required by these protocols and executing them.

This chapter first gives in Section 6.2 a world-tour overview of the existing types of EHR architectures. In Section 6.3, it describes the key hardware and software components allowing to make full-fledged personal data servers out of tokens, based on which it then proposes in Section 6.4 a realistic approach complementary to existing EHR solutions. We take advantage of this proposal to discuss the practical issues related to the privacy-preserving data publishing protocols proposed in this thesis. Finally, we synthesize and discuss the chapter in Section 6.5.

6.2 Overview of Electronic Health Record Projects

An Electronic Health Record (EHR) system is a collection of electronic patient folders, each containing the complete medical history of an individual, managed and consulted by authorized health professionals across several organizations [NAHIT08]. Building an EHR system requires interconnecting various heterogeneous health information systems of disparate organizations in order to aggregate the medical data they maintain separately (e.g., hospitals, practitioners and pharmacists data related to a same individual). Hence, the first challenge tackled by EHR programs is providing interoperability between heterogeneous systems. As pointed out in the introduction of the chapter, ensuring data availability even in disconnected environments and enforcing data security are two additional and mandatory challenges. The following presents a state of the art on these three challenges, from the most centralized approaches to the most decentralized ones.

6.2.1 Centralized Records

The most integrated approach seeks to gather all the medical records related to the same individual into a centralized healthcare folder.

In the USA, some private organizations had already felt the need to aggregate all their patients' data in a single folder before the launch of regional or national initiatives. For example, the Veteran Administration Medical Center developed the VistA system

[Brown03], a Health Information System (HIS) enabling health centers with VistA to share their patients' data.

The French national program Dossier Medical Personnel[24] (DMP) aims also at centralizing healthcare folders hosted by selected Database Service Providers.

Systems like Google Health™[25] and Microsoft's HealthVault™[26] propose to individuals to centralize their health records on their own initiative. Both load medical data directly from the patient's health centers, offer practical tools for individuals (e.g., drug interactions, hospitals searches), and can provide a controlled access to the record to a selected set of persons. Both are free and the users are simply asked to trust their privacy policy. Note that due to a low user adoption, Google™plans to stop Google Health™in January 2012 [Google11].

6.2.2 Centralized Index

The second approach strengthens the integration thanks to centralized indexes and/or data summaries.

The National Health Society in the United Kingdom has launched the Care Record Service (CRS) project[27]. First, CRS aims at linking together the existing medical records of an individual, thus constituting a virtual unique health folder. Navigating between the records of an individual and gathering detailed data will be easier; furthermore, by sharing data across records, duplication of - e.g., administrative - data will be useless. Second, CRS aims at storing summaries of detailed data on the *Spine*, an already existing central system currently in charge of delivering health related services (e.g., ePresciptions, eReservations). The Secondary Uses Service (SUS) of CRS will use summarized data to draw reports and analysis about collected care information.

With its two-level functional architecture, the Diraya project from Andalusia[28] is similar to the UK's CRS project. First, detailed data in medical records are kept where they are produced (e.g., hospitals) and the central Diraya system indexes them. Second, Diraya centralizes what is called the *main data*, that is the data most frequently accessed.

In the Netherlands, the National Healthcare Information Hub project (LSP in Dutch), led by Nictiz[29] is basically a central index storing the location of every individual's medical record.

[24] http://www.d-m-p.org/
[25] http://www.google.com/health/
[26] http://www.healthvault.com/
[27] http://www.connectingforhealth.nhs.uk/
[28] http://www.juntadeandalucia.es/
[29] http://www.nictiz.eu/

The Austrian ELGA initiative [Husek08] is similar to the LSP project.

The Canadian national program Infoway-Inforoute[30] funds provincial EHR projects, most of these focusing on interoperability between care organizations. For example, the Alberta Netcare EHR[31] centralizes regionally the patients' summary data; the Ontario EMRXtra project extends the medical records to local pharmacists by using an encrypted website[32]; the Yukon Telehealth project makes local medical records accessible to remote specialist practitioners.

The Australian EHR project[33] is not yet clearly defined, but seems also to direct towards a centralized summary health folder that aims to enhance communication across health organizations.

6.2.3 Interconnected Autonomous Servers

The third approach consists in interconnecting existing autonomous systems in a wider infrastructure.

The Danish Healthcare Data Network[34] [Petersen06] is representative of this category. It connects the already secure intranets of care organizations via VPNs over the Internet, progressively from organizations to counties, counties to regions, and regions to nation. The Danish EHR effort has mainly consisted in defining a common data model representing clinical data.

The USA have adopted a federal approach to build the EHR. At the region scale, Regional Health Information Organizations (RHIOs) are multi-stakeholder organizations that enable the exchange of health information between local health organizations (e.g., CalRHIO for the Californian RHIO). At the nation scale, the Nationwide Health Information Network (NHIN) project[35], supervised by the Office of the National Coordinator for Health IT (ONC), aims at enabling secure health information exchange across the USA by using RHIOs as regional building blocks. The NHIN will be a network of health networks built over the Internet.

[30] http://www.infoway-inforoute.ca/
[31] http://www.albertanetcare.ca/9.htm/
[32] http://www.ghc.on.ca/health/publications.html?ID=121
[33] http://www.nehta.gov.au
[34] The Danish health minister stopped the EHR effort in 2006 to make the EHR centrally controllable [Dahl07].
[35] http://www.hhs.gov/healthit

6.2.4 Current Token-Based Approaches: Disconnected Accesses

All the systems mentioned above provide a 24h/7day a week availability assuming the servers are active and an Internet connection can be established to reach them. This is unfortunately not the case in every place and situation, introducing the need for disconnected accesses to healthcare folders.

In the United Kingdom, the Health eCard [36] is a private initiative that proposes to store encrypted copies of full medical records in specifically designed smart cards, making patients' health data available in disconnected situations (e.g., emergency situations, home consultation).

Taiwan has launched in 2001 a project to replace the traditional paper health cards by smart cards [Alliance03]. Smart cards are used exactly as paper cards were used. They permanently store administrative personal and summary health data, and temporarily store the medical data related to the last six visits. Every six visits, the temporary medical data are uploaded into the Taiwanese health infrastructure. The smart card health project is seamlessly integrated with the previous health infrastructure, providing a strong patient authentication and a paperless data management.

The German organization Gematik[37] leads the eGK, an ambitious project mixing a traditional infrastructure with smart cards in order to tackle connected and disconnected situations [Alliance06]. Both patients and professionals are equipped with a smart card, patient smart cards storing EHRs while professional smart cards are being used for strong authentication, digital signature and encryption/decryption of documents. The infrastructure holds a centralized copy of the EHRs, accessible through the Internet. This project is still at a preliminary stage.

In the USA, many private initiatives issued by care centers tackle the disconnected access requirement [Alliance06], e.g., the University of Pittsburgh Medical Center Health Passport Project (HPP), the Florida eLife-Card, Queens Health Network, Mount Sinai Medical Center Personal Health Card. All of them store a copy of critical health information encrypted on a smart card to make it available in case of emergency.

6.2.5 Promoting a Token-Based Approach for Tackling Security Issues

Strong authentication is usually required to connect to EHR servers. Health professionals authenticate with a smart-card (e.g., the CRS in UK, the LSP in the Netherlands), as well as patients accessing to their medical folder (e.g., the Diraya initiative in

[36] http://www.healthecard.co.uk/
[37] http://www.gematik.de

Andalusia). In addition, communication channels can be protected by cryptographic techniques, based on protocols such as TLS [Dierks08], enabling entities to securely exchange messages (i.e., encryption, integrity protection, non repudiation of messages), and security measures are implemented on central servers. However, this is not sufficient to put trust in the system.

The suspicion is fueled by computer security surveys pointing out the vulnerability of database servers against external and internal attacks [Gordon06]. Database systems are identified as the primary target of computer criminality [Gordon06], and even the most well defended servers, including those of Pentagon [Sevastopulo07, Liebert08], FBI [Weiss07] and NASA [Rosencrance03], have been successfully attacked. In addition, nearly half of the attacks [Gordon06] come from the inside (employees) of the companies or organizations. In addition, there are many examples where negligence leads to personal data leakages. To cite a few, thousands of Medicare and Medicaid patients in eight states have been lost in a HCA regional office [IT06] and Hospitals County published by accident medical records on the web [IT08, WFTV08] including doctors' notes, diagnoses, medical procedures and possibly names and ages of patients. A recent study shows that 81% of US firms declare loosing employees laptops with sensitive data [Rosencrance06]. Data loss is so frequent that a research project called DataLossDB[38] has been created to report such incidents.

In practice, EHRs are thus very difficult to protect. This legitimates the reserves expressed by both practitioners and patients about EHR programs [Rose08, Gratzel08]. In the Netherlands, privacy and access concerns are major arguments for the postponement of the national EHR [Bos09]. In particular, the lack of security measures limiting data access for service providers and the loss of control on their own data has been identified as a main reason for citizens to opt-out of the system. Only medical records stored in personal and secure hardware such as smart-cards can benefit from true privacy enforcement. The above token-based approaches only give a beginning of answer to this challenge: (1) the storage capacity of the smart-cards used by current projects (e.g., from KB to MB) is too low to store a complete EHR, limiting the data availability in disconnected situations, (2) their low connectivity makes the hosted data seldom available, and (3) their portable nature makes them likely to be lost or broken.

[38] http://datalossdb.org/

6.3 A Secure and Portable Medical Folder

Researches conducted in the PlugDB Project[39] resulted in a lightweight Database Management System (DBMS) embedded in a token. Roughly speaking, a token combines a secure micro-controller (similar to a smart card chip) with a large external Flash memory (Gigabyte sized) on a USB key form factor [Eurosmart08]. A token can host a large volume of on-board data and run on-board code with proven security properties thanks to its tamper-resistant hardware and a certified operating system [Eurosmart08]. The main target of the PlugDB technology is the management of secure and portable personal folder. Healthcare folders are a very good representative of large personal folders where security and portability are highly required.

Compared to smart cards used in other EHR projects (see Section 6.2), the storage capacity of a token is roughly four orders of magnitude higher. Henceforth, this makes sense to embed the whole patient folder in her token and make it available in disconnected mode. In addition to the data, a complete chain of software is embedded in the token micro-controller: (1) a Web server, (2) servlets implementing the application, (3) a JDBC bridge, and (4) a DBMS engine managing the on-board database and enforcing access control. Hence, the token along with its embedded database and software can be seen as a full-fledged server accessible through any web browser running on any device equipped with a USB port (e.g., laptop, tablet-PC, PDA and even cell-phone). Compared to a regular server, the token server is personal, pluggable, does not require any network connection, and provides unprecedented security guarantees.

The specific hardware architecture of the token introduces however many technical challenges. We detail below the most important ones.

6.3.1 Hardware and Operating System

A token combines in the same hardware platform a secure chip and a mass storage NAND FLASH memory (several Gigabytes soon). The secure chip is of the smart card type, with a 32 bit RISC CPU clocked at about 50 MHz, memory modules composed of ROM, tens of KB of static RAM, a small quantity of internal stable storage (NOR FLASH) and security modules. The mass storage NAND FLASH memory is outside the secure chip, connected to it by a bus, and does not benefit from the chip hardware protection.

Gemalto, the smart card world leader, has developed an experimental token platform. This platform includes a new multi-tasking operating system allowing the devel-

[39] PlugDB (http://www-smis.inria.fr/~DMSP) is a project funded by ANR, the French National Research Agency.

opment of Web applications based on JAVA and Servlet technology, and thus offering a standardized means to integrate services and embedded Web applications into the token. The operating system supports natively: the USB 2.0 protocol and the Internet protocol IP for communicating with the external world, multi-threaded Java applications, cryptographic primitives (some of which implemented in hardware), memory management and garbage collection, servlet management and Web server. For more technical details on the hardware platform and the operating system, we refer the reader to http://www-smis.inria.fr/~DMSP.

6.3.2 Embedded Database System

DBMS designers have produced light versions of their systems for personal assistants (e.g. Oracle-lite, DB2 everyplace, SQLServer for Window CE) but they never addressed the more complex problem of embedding a DBMS in a chip. Initial attempts towards a smart card DBMS were ISOL's SQL Java Machine [Carrasco99], the ISO standard SCQL [ISO99], and the MasterCard Open Data Storage [Mas02]. All these proposals concerned traditional smart cards with few resources and therefore proposed basic data management functionalities (close to sequential files). Managing embedded medical folders requires much more powerful storage, indexation, access control and query capabilities. PicoDBMS was the first full fledged relational DBMS embedded in a smart card [Pucheral01] and was implemented on top of Gemalto's smart card prototypes [Anciaux01]. PicoDBMS has been designed for managing databases stored in a (MB sized) EEPROM stable memory integrated in the secure chip and protected by the tamper-resistance of the chip.

The token framework introduces important new challenges [Anciaux07b]:

1. How to support complex queries over a large on-board database (Gigabyte sized) with very little RAM (a few Kilobytes)?
2. How to organize the data storage and the indexes with an acceptable insert/update time considering the peculiarities of NAND Flash memory (fast reads, costly writes, block-erase-before-page-rewrite constraint)?
3. How to protect the on-board database against confidentiality and integrity attacks (the external Flash being not hardware protected) while keeping acceptable query performance?

The token architecture and the organization of the embedded software components are illustrated in Figure 6.1. The on-board code and sensitive data (e.g., cryptographic keys) reside in the secure chip, patient's data reside in the insecure external memory

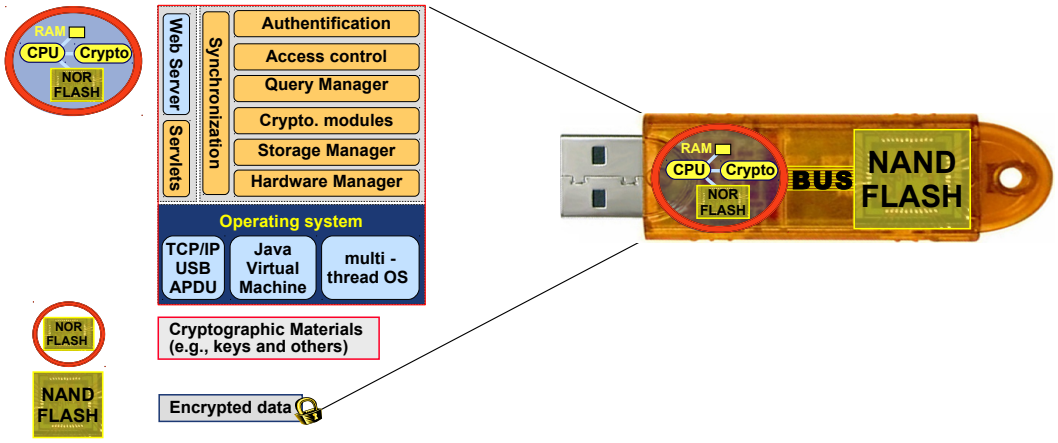


Figure 6.1: Internal Architecture of a Secure Portable Token

(previously encrypted by the secure execution environment). We detail below the components related to the technical challenges mentioned above.

The *Query Manager* is in charge of parsing the incoming database queries, building an optimal query execution plan and executing it. This module must consider peculiar execution strategies to answer complex SQL queries over a large quantity of data with little RAM (challenge 1). To tackle this challenge, [Anciaux07a] designed a massive indexing scheme which allows processing complex queries while consuming as little RAM as possible and still exhibiting acceptable performance. The idea is to combine in the same indexing model generalized join indices and multi-table selection indices in such a way that any combination of selection and join predicates can be evaluated by set operations over lists of sorted tuple identifiers. The operator library (algorithms for the operators of the relational algebra, e.g., select, project, join and aggregate) and the execution engine integrate those techniques.

The *Storage Manager* on which the query manager relies to access the database content (index and tables) is directly concerned with challenge 2. Indeed, the proposed massive indexation scheme causes a difficult problem in terms of Flash updates, due to the severe read/write constraints of NAND Flash (rewriting NAND Flash pages is a very costly operation). Therefore, [Yin09] designed a structure which manages data and index keys sequentially so that the number of rewrites in Flash is minimized. The use of summarization structures based on bloom filters [Bloom70] and vertical partitioning reduces the cost of index lookups. These additional structures are also managed in sequence. A first implementation of this principle has been patented jointly by INRIA and Gemalto [Pucheral07] and is integrated in the current DBMS prototype.

The *Hardware Manager* embeds the methods for accessing the different memory modules of the token. It includes techniques associated with challenge 3 to protect the confidentiality and the integrity of the data, in an efficient way with respect to DBMS access patterns. Indeed, the massive indexation technique leads to numerous, random and fine grain accesses to raw data. [Anciaux06] conducted preliminary studies, combining encryption, hashing and timestamping techniques with query execution techniques in order to satisfy three conflicting objectives: efficiency, high security and compliance with the chip hardware resources.

6.3.3 Data Availability and Security

Any terminal equipped with an USB port and a Web Browser can interact with the token and get the data it is granted access to. Hence, when no Internet connection is available (e.g., emergency situations, home intervention, remote server breakdown) tokens guarantee patients' data availability. Furthermore, local connections to tokens do not suffer from unpredictable performance due to overloaded remote servers or low quality connections: the embedded server is mono-user and USB-2 communication throughput is guaranteed.

In terms of security, patient's data resides in the external NAND Flash memory. As stated in section 6.3.1, this memory is not hardware protected; its content must be encrypted to prevent confidentiality attacks and hashed to prevent integrity attacks. The cryptographic keys serving this purpose reside in the NOR Flash memory and are protected by the tamper-resistance of the secure chip. The encryption/decryption/hashing processes physically take place in the secure chip and are similarly hardware protected (see Figure 6.1). More generally, the complete software chain (web server, servlets, DBMS) runs in the secure chip and benefits from its tamper-resistance. Hence, the authentication, access control, and query steps are all hardware protected. The security of the complete architecture thereby relies on the tamper-resistance of the secure chip. The security of the hardware platform and of the embedded code is under certification with the goal to reach the highest security level (EAL4+), usually required for smart card chips used in the medical domain. This makes attacks highly improbable and extremely costly to conduct. Considering that the security of a system lies in the fact that the cost to conduct an attack outweighs its benefit, the security of our architecture is reinforced by the extreme cost of attacks and their small benefit (disclosure of a single patient's folder).

6.4 Data Sharing Issues

Tokens provide a secure disconnected access to patients' health data. This section focuses on making patient's healthcare folder seamlessly available without the patient's token for answering questions like: (1) how can the family doctor express her opinion about an emergency situation without the patient's token on hand, or (2) how can survey institutes draw useful data analysis without having access to the patients' raw data? We introduce a supporting central server in the architecture as a mean to obtain the required data availability. This must be achieved without losing the benefits of the token in terms of security and control by its owner: sensitive information must not be unveiled to the supporting server. Consequently, sensitive data must be stored encrypted on the server storage media and must be encrypted and decrypted only in a token's secure execution environment.

Protocols such as TLS [Dierks08] enable secure communications between tokens and supporting servers. TLS relies on a certificate authority to emit trusted certificates linking an identity to a public key. The professionals, patients, and supporting servers safely communicate after having exchanged and checked their respective certificates. Similarly to the cryptographic material used by tokens, certificates are inserted in the tokens' secure internal memory at the beginning of their life cycle, before being delivered to its owner (see Section 6.4.4 for a discussion about setting the parameters). The server is in charge of his own cryptographic material's security and durability. Tokens, however, are personal devices; they may be lost, or broken. Their certificates and pairs of (public key, private key) must be made durable by being replicated in a trusted storage area that provides strong guarantees of availability. For simplicity, we let aside the design of protocols among secure tokens that could endorse this role, and assume the existence of a trusted third party.

In this section, we first propose a classification of individual data driven by the individual's own wish for privacy. Second, we outline a simple synchronisation protocol in charge of coping with highly disconnected tokens. We then depict the resulting token-centered architecture. Finally, we discuss the practical issues related to the privacy-preserving data publishing protocols proposed in this thesis.

6.4.1 A Data Classification Driven by Privacy Concerns

In the following, we loosely call *data* any piece of information without assuming anything about their actual nature (e.g., relational attribute, image, object). We propose to classify data according to their level of sensitivity. Note that though the sensitivity classes are fixed, the actual classification mapping a given data to a class is highly subjective, so may vary from a patient to another.

At the two extremes of the privacy spectrum lie the classes containing *secret data* (the most sensitive) and *regular data* (the least sensitive).

- Secret data are considered so sensitive (e.g., psychological analysis) that they remain sealed into the patient's token. The durability of secret data is not guaranteed by the architecture; if desired, it must be enforced by the patient itself (e.g., replicated into another token physically stored in a secure place).

- Opposite to secret data, regular data are non sensitive pieces of information; as such, they can be stored in the clear on the supporting server. Regular data is protected by the security policy enforced by the server and can be accessed online by any practitioner having the required privileges. The access control policies enforced by the server and the token are assumed to be identical, but since supporting servers are not tamper resistant, regular data could be accessed on the server without prior patient knowledge. We detail below how regular data can be defined.

Complementary to the secret and regular data classes, the *confined data* and *anonymous data* classes allow to trade privacy and utility.

- Confined data are these pieces of information too sensitive to be managed as regular data but for which online availability and/or durability is mandatory for care practice (e.g., HIV diagnosis, chemotherapy medication, MRI image). Confined data is replicated, encrypted, on the server, but without their encryption key. To ensure online availability, encryption keys are shared among a set of tokens selected by the patient (e.g., the family doctor's token, the token of a trusted specialized physician); these tokens make up the *trusted circle* of the patient. Members of the trusted circle access encrypted confined data on the server (after having proven their membership into the patient's trusted circle through the *proof of trust*, e.g., a signature emitted by the patient's token) and decrypt it locally in the token's secure execution environment. The durability of encrypted confined data is guaranteed by the server similarly to the clear-text regular data. Encryption keys however must be made durable through other means (e.g., a trusted third party, a synchronization protocol among members of the trusted circle).

- Finally, anonymous data are pieces of information that can be externalized to contribute to privacy-preserving data publishing surveys (e.g., an epidemiological study), provided they will be properly sanitized.

Although we do not formulate any assumption concerning the data classification process, we emphasize that it should be done carefully in order to avoid prying eyes to infer secret or confined data based on regular or anonymous data. For illustration purposes, let describe briefly a possible classification process. First, the pieces of information contained in the secret and confined classes are fixed by the patient. The patient's physician and normative groups (e.g., consisting of physicians, patients) can favor homogeneity by helping patients in reaching enlightened decisions. Finally, an oracle - executed on the patient's token - determines which pieces of information can be considered as regular or anonymous data without endangering the privacy of any secret or confined data. Note that for the sake of privacy, the patient is not able to add or remove pieces of information to the regular or anonymous classes by itself.

6.4.2 Synchronization

Replicating data on the central server provides availability and durability, but raises a synchronization problem. When the server and a token are directly connected with each other, traditional synchronization methods apply. However, it may arrive that a token seldom connects directly to the central server (e.g. a patient who seldom leaves home). In this case, tokens of health professionals endorse the role of proxies, carrying encrypted synchronization messages between patients's tokens and the central server.

Health professionals carry encrypted synchronization messages from patients' tokens to the central server when they visit the patients. During the visit, the professional may insert new data in the patient's token. At the end of the visit, regular and confined data not present yet in the server are copied into the professional's token. The central server is refreshed every time a professional connects to it. A similar protocol occurs between the professional's token and the publisher (possibly through the supporting server) for collecting freshly created anonymous data formatted according to the standard PPDP format. Conversely, the professional's token carries encrypted data newly created at the server in order to refresh the patient's token. This situation occurs when external entities produce medical data directly on the central server, e.g., a laboratory producing examination results. However, such *external data* cannot be produced in the clear and it is not yet classified by the patient. To circumvent this problem, external entities must encrypt external data based on the patient's public key before publishing them on the central server. At synchronization time, the patient will be able to decrypt this data and classify it (obviously not as a secret data, except if the supporting server is assumed to be honest-but-curious such that it erases properly[40] the freshly classified data).

[40] Based on, e.g., secure deletion techniques [Stahlberg07].

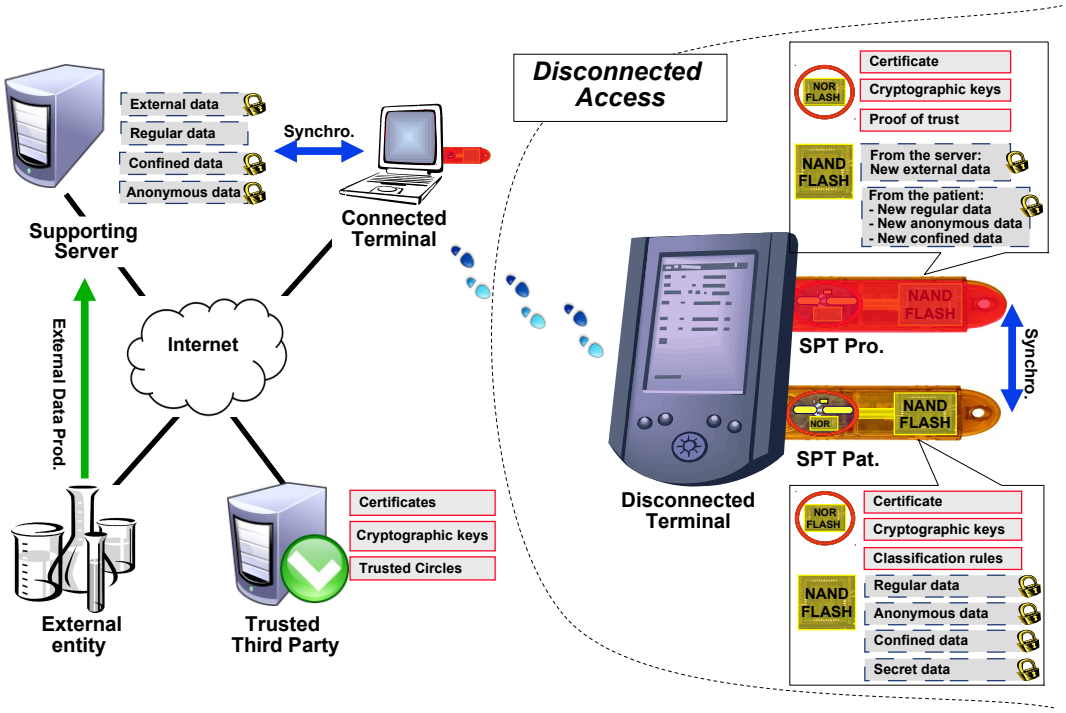


Figure 6.2: Architecture Supporting the Data Classification and the Synchronization Protocol

6.4.3 Supporting Architecture

Figure 6.2 depicts the architecture supporting the data classification and the synchronization protocol, showing where information resides. Data located in *solid* and *red* rectangles reside in a trusted storage (either the token's internal memory or the trusted third party) contrary to data located in *dashed* and *blue* rectangles. Data aside yellow locks is encrypted. This architecture provides stronger privacy preservation guarantees than any traditional EHR. Attacks conducted at the supporting server (bypassing the traditional security measures) can only reveal regular data, secret data being absent from the server, and confined and anonymous data being encrypted based on cryptographic material to which the supporting server does not have access. Attacks conducted over a patient's token are made highly difficult thanks to the tamper resistance of the secure micro-controller, and the encryption of data stored in the external flash memory.

6.4.4 Practical Issues of the PPDP Protocols

The following discusses the practical aspects related to the implementation of the privacy-preserving data publishing protocols proposed in this thesis.

Setting the Parameters

The privacy-preserving data publishing protocols proposed in this thesis assume that local parameters are pre-installed within tokens before running any protocol instance. We describe here a possible way to set them up.

Behind the term local parameters, various types of parameters are hiding, possibly set differently. There are *user-defined parameters*. The approach promoted in this thesis is individual centric and naturally allows the individuals to specify their own privacy preferences through their tokens. There are also *dynamic parameters*, that may vary with each running instance (e.g., the model, the algorithm, the privacy parameters enforced). There are finally the *roots of trust parameters*: the parameters necessary to the protocols execution and based on which the dynamic parameters can securely be set.

User-Defined Parameters. Tokens are personal devices; each token, combined with an intuitive web-based GUI, allows its owner to define naturally his own privacy requirements (e.g., expressed as a coarse privacy range such as "Altruistic" or "Armoured"). Privacy requirements typically specify the privacy models trusted by the user (e.g., l-diversity, not k-anonymity), the minimal privacy parameters (e.g., the minimal l and k values), the minimal detection probability to enforce (for PROTOGEN$_{(wm)}$). A token participates in the collection phase of a protocol's running instance only if the privacy parameters announced by the publisher fit its owner-defined parameters. The user-defined privacy level is compared to the protocol's running instance's privacy level: if the latter is - loosely speaking - higher than the former, then the token participates to the collection phase; otherwise it does not.

Dynamic Parameters. A running instance of a sanitization protocol is based on various parameters, e.g., a privacy model, a privacy algorithm, the corresponding privacy parameters, a dataset cardinality. These parameters must be announced by the publisher to the tokens that connect during the instance's collection phase, such that they decide whether to participate or not in the release based on their own user-defined parameters. These parameters are *dynamic* in that they vary from one running instance to another. However, they do not change during the run of their corresponding instance (tokens can check that they are invariant).

Roots of Trust Parameters. Three *roots of trust parameters* allow the initialization and the execution of protocols. The primitive cryptographic material enabling secure communications between both tokens alone, and tokens and the publisher, is the first root of trust. The `Invariance` parameters (e.g., the designated population for META-PROTOCOL$_{(wm)}$ - e.g., set to be a subset of the tokens delivered to health professionals), allowing tokens's choices to sit on a stable basis, are the second root of trust. Finally, the execution sequences of the various protocols form the third root of trust. These parameters cannot be decided by tokens alone: they have to be installed within the secure enclosure of tokens by a trusted external entity (e.g., the tokens's manufacturer). Based on them, tokens communicate securely, make invariant the dynamic parameters, and execute securely the various execution steps of running instances.

Illustrative Scenario. Let concretize these parameters through a simple example based on META-PROTOCOL$_{(wm)}$. Alice, our geeky eighty-years old woman, has just received her secure portable token in charge of storing her health folder. During her first medical visit, her physician helps her classifying her medical data in the secret, confined, regular, and anonymous classes, based on the standard classification proposed by the well-known *MyMedicalToken* normative group. She also chooses the privacy level to be enforced upon her anonymous data. The security and privacy guarantees of her token and of privacy-preserving data publishing schemes, as well as her willingness to be beneficial to others, make her choose the "Altruistic" privacy level, allowing her token to contribute to various sanitized releases. Let assume for simplicity that her anonymous data consists of a single record.

The day arrives when her token is contacted by a statistical agency for launching an epidemiological survey. Alice's token and the statistical agency authenticate together based on the cryptographic material pre-installed within the token. The statistical agency then announces the schema of collected data, and the privacy level that will be enforced by the sanitized release: the recipients: an academic group of epidemiologists, the privacy model: k-ANONYMITY where $k = 100$, and the privacy algorithm: MONDRIAN based on order-preserving encrypted quasi-identifiers. Alice's token checks the certificate binding the group of epidemiologists to both the agency and the privacy parameters based on the crypto-material pre-installed within the token by the manufacturer: the token knows now that the agency is honest with respect to the recipient of the release. It also checks that these parameters have been previously made invariant (based on the pre-installed crypto-material and designated population's size). Knowing now that the agency's announce is reliable, it sends Alice's anonymous data following the META-PROTOCOL$_{(wm)}$'s tuple format, having augmented the auxiliary part with a signature binding the tuple to the agency's announce. The token that will sanitize Alice's tuple will be able to check that it was indeed produced for the actual

privacy parameters.

Single Execution of a Single Protocol

Now that we have securely set the various parameters, let us discuss the practical aspects related to the protocol's execution. For the presentation's clarity, we focus for the moment on a single execution of a single protocol.

Collected sample. There exists an inherent statistical bias in the processing model: tokens that do not connect during the collection phase do not appear in the sanitized release (e.g., healthy patients). However, the bias is not due to the token based approach but to the underlying behaviour of the population surveyed (individuals unavailable with tokens would still be unavailable without tokens). This bias, called *sampling bias*, is well-known and traditionally corrected through various techniques [Ards98, Fadem08, Cortes08].

Tokens's Connections. No assumption is made concerning the number of tokens concurrently connected, or the connections times or number of a single given token. There only needs to be enough tokens's connections to perform the complete execution sequence. This makes the protocols proposed in this thesis adaptable to a wide variety of applicative situations.

Single Execution of Multiple Protocols

Different recipients need different sanitized releases. However, naively executing several protocols on the same dataset by the same publisher could lead to severe data breaches. Indeed, the amount of data leaked by a joint analysis of the information voluntarily disclosed for several protocols may thwart the privacy of all protocols. Consider for illustration purposes the execution of both a generalization-based and the BUCKETIZATION algorithms. The former could be based on the disclosure of the record's quasi-identifier value in clear text, while the latter on the disclosure of the record's sensitive value, in clear text too. Taken independently, the disclosed information do not endanger the individual's privacy; taken together, they definitely do.

The *separation of duty* paradigm [Saltzer75] (also called *separation of privileges* or the *two-man rule*) is a fundamental principle in computer security. It can preclude such joint analysis by requiring each protocol to be under the responsibility of a distinct entity. A straightforward way to concretize this principle in our context is to require that each protocol be executed by a distinct organization. This may be however not always practical. Executing several protocols by the same organization, within which

security policies enforcing the separation of duty principle (e.g., [Li07b, Li08]) are implemented, is another alternative.

Continuous Data Publishing Problem

How can an evolving dataset be sanitized several times along its evolution by the same algorithm, such that a joint analysis of the various releases does not thwart the privacy guarantees? The models and algorithms designed to answer this question in a centralized context require keeping (a partial) track of the previous sanitized releases [Byun06, Xiao07, Bu08, Fung08, Byun09]. We let to future work the investigation of this *continuous data publishing* problem within this thesis's approach (see the perspectives drawn in Section 7.3).

6.5 Synthesis

EHR projects are being launched in most developed countries. The benefits provided by centralizing the healthcare information in database servers in terms of information quality, availability and protection against failure are unquestionable. Yet, patients are reluctant to abandon the control over highly sensitive data (e.g., data revealing a severe or (considered) shameful disease) to a distant server. In addition, the access to the folder is conditioned by the existence of a high speed and secure Internet connection at any place and any time. This chapter gives the control back to the patient over his medical history by capitalizing on tokens. We have shown how this device can complement a traditional EHR server (1) to protect and share highly sensitive data among trusted parties, (2) to provide a seamless access to the data even in disconnected mode, and (3) to enable global computations, like privacy-preserving data publishing, that do not rely primarily on a trusted central server. A discussion about the practical aspects of setting and executing the privacy-preserving data publishing protocols proposed in this thesis has closed the chapter.

Chapter 7

Conclusion and Perspectives

Summary: *Time has come to close this manuscript. This chapter synthesizes the work conducted in this thesis and highlights several directions for future works. A first category of future works considers the internals of the* Meta-Protocol*: exploring the blank space defined by the* ***Genericity*** *and* ***Generality*** *dimensions in the context of decentralized approaches, adapting the safety properties' implementations to cope with a publisher colluding with a set of tokens, and designing new privacy models and algorithms especially for the* Asymmetric *architecture. A second category considers other privacy-preserving data publishing contexts: data types other than tabular, continuous data publishing, and interactive data publishing. Interestingly, continuous and interactive data publishing share the similar challenge of handling a certain form of* history*. In closing, we conclude by analyzing how the findings of this thesis could serve as a basis for tackling the distributed evaluation of global functions other than privacy-preserving data publishing.*

Contents

7.1 Introduction

THE three quarters of the ever-growing digital world is generated by individuals. On the one hand, the ability to analyze such a wealth of data carries many societal promises, notably concerning human health. On the other hand, individuals' privacy is more than ever at stake. Privacy-preserving data publishing is an attempt to address jointly both aspects: *how can personal data be published for analysis purposes without endangering the individuals' privacy?* The answer is not trivial. So far, most works have focused on privacy models and algorithms to be executed by the *publisher*, a central server *presumably* trusted. A quick tour on the news however highlights the inadequacy of this trust assumption in practice. Moreover, permissive legislations favor data usage at the expense of individuals' privacy. Current practices are a major source of vulnerabilities.

In this thesis, we have questioned the trusted publisher assumption, building on an emerging type of device called secure portable token. By combining a tamper-resistant micro-controller to a large secondary storage memory in a portable form factor, tokens are the basis for building full-fledged secure personal data servers. This thesis has suggested to reach the objectives of **Decentralization**, **Genericity**, and **Generality** in the context of non-interactive privacy-preserving data publishing, focusing especially on the centralized publishing family of models and algorithms. In this context, some approaches have questioned the trust put in the publisher - but in a way that severely limits their scope: to the best of our knowledge, no existing approach reaches simultaneously the aforementioned objectives. Figure 7.1 roughly positions the main contribution of this thesis - namely the META-PROTOCOL - with respect to the (few) other decentralized approaches to centralized publishing - namely generic secure multi-party computation and specific secure multi-party computation dedicated to centralized publishing. The x-axis positions the approaches with respect to the (qualitative) level of **Generality** reached (i.e., the required availability and the incurred cost), and the y-axis with respect to the (qualitative also) level of **Genericity** reached.

The approach promoted in this thesis is part of the larger *personal data server* vision. Described in [Allard10a], it draws the lines of an encompassive individual-centric architecture that aims at enabling at the same time powerful personal data applications as well as a friendly individual control over her data coming with tangible enforcement guarantees. The architecture is based on the *personal data server*, a token embedding a suite of software allowing it (1) to provide the main functionalities of a database engine (see Section 6.3 above), (2) to be interoperable (acquire personal data from existing data servers, securely share data with other personal data servers), and (3) to enforce the hippocratic privacy principles [Agrawal02] (e.g., consent, limited collection, limited

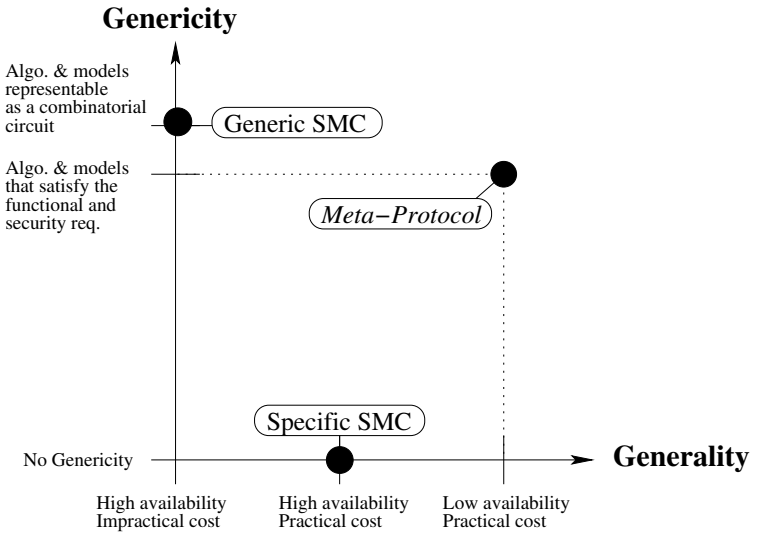


Figure 7.1: Decentralized Approaches to Centralized Publishing Algorithms

retention, audit), all without disrupting the token's portability and security features. Though not circumscribed to the above vision, the privacy-preserving data publishing protocols proposed in this thesis restore privacy-preserving data publishing abilities within the personal data server architecture.

This chapter concludes the thesis. We synthesize the work conducted, and close the manuscript by opening exciting research perspectives. We present them in the order of increasing scope: (1) concerning the META-PROTOCOL, (2) concerning privacy-preserving data publishing models and algorithms, and (3) concerning *global function evaluation* in a token-based architecture.

7.2 Synthesis

7.2.1 Overview

The journey into the major centralized publishing models and algorithms (Section 2.2) emphasizes the need for various models and algorithms compatible with the variety of practical settings. In short, it illustrates the absence of a *one-size-fits-all* approach to privacy-preserving data publishing. Based on this statement, we extracted the **collection**, **construction**, and **sanitization** phases common to centralized publishing algorithms (Section 3.1) and remodeled them within the ASYMMETRIC architecture such that they form a robust ground for reaching the **Decentralization**, **Genericity**, and **Generality** objectives. The ASYMMETRIC architecture is made, on the one hand, of

trustworthy tokens exhibiting low availability and modest computing resources, and, on the other hand, of an untrusted central server, the publisher, that exhibits high availability and high computing resources. The basis for achieving the **Genericity** objective is to delegate the **construction** phase to the publisher and to parallelize the **collection** and **sanitization** phases over the tokens. Since the (untrusted) publisher is in charge of a part of the execution sequence, it is crucial to guarantee the security of the sequence. We have considered two (well-known) types of adversarial behavior: the passive honest-but-curious attacker, and the weakly-malicious attacker (active only if he is not convicted as an adversary and if he can produce a correct result). The correctness and security models of the protocols are based on the strong computational indistinguishability requirement, well-known in secure multi-party computation approaches.

We achieved the objectives of this thesis in two steps. During the first step, we simplified the problem by (1) reducing the **Genericity** objective to generalization-based algorithms for enforcing the k-Anonymity model and (2) limiting the execution sequence of the protocols to a single step per phase. This resulted in the Protogen suite of protocols. The second step consisted in generalizing the findings of the first step to reach the full **Genericity** objective without restricting the execution sequence of the protocol. This resulted in the Meta-Protocol suite of protocols. The Protogen and Meta-Protocol suites were designed in similar molds. We started to tackle the honest-but-curious attack model, thereby laying the foundations of each suite, i.e., the basic execution sequence free from data leaks. We then tackled the weakly-malicious attack model by precluding any tampering action upon which the adversary could base his attacks. The protection against both passive and active attacks is guaranteed by a small set of safety properties to be asserted during the execution sequence. We designed the safety properties in two stages (similarly to cryptographic schemes proposals): a definition stage - answering : *"what must be checked for preventing tampering actions?"*, and a construction stage - answering : *"how can the checks be performed by proof-of-concept implementations?"*. The third contribution of this thesis is the informal discussion of the practical aspects of the suites of protocols. We based this discussion on the implementation of an electronic health record system where patients are equipped with tokens and use them as their primary data servers.

We synthesize below each of the contributions.

7.2.2 The Protogen Suite

The Protogen suite of protocols focuses on generalization-based algorithms for enforcing the k-Anonymity privacy model. The notion of *equivalence class* is central in Protogen. The honest-but-curious version of the protocol consists in the following:

(1) during the **collection** phase, tokens send their quasi-identifier value in the clear and their sensitive value encrypted, (2) during the **construction** phase, the publisher forms the equivalence classes by means of the global recoding generalization-based algorithm of his choice, and (3) during the **sanitization** phase, each connecting token downloads the sensitive values contained by an equivalence class and returns them, decrypted and shuffled, to the publisher. We demonstrated the correctness and security of this execution sequence against the honest-but-curious publisher. However, k-ANONYMITY is endangered when considering the weakly-malicious attack model. Indeed, the publisher may form equivalence classes containing, e.g., less than k records, or sets of records that overlap such that differentiating the sensitive values returned permits an accurate reconstruction of the initial raw records. We showed that embedding the following safety properties in the execution sequence prevents any tampering action:

- `Origin` precludes the publisher to *forge* tuples;
- `Identifier Unicity` precludes the publisher to *copy* a tuple in its own equivalence class;
- `Membership` and `Mutual Exclusion` preclude the publisher to *copy* a tuple across equivalence classes;
- `Invariance` precludes the publisher to associate a given generalization node to several sets of tuples;
- `Safe Rules` precludes the publisher to form an equivalence class that contains less than k tuples;

Simple implementations of these safety properties were proposed with the single-step-per-phase constraint in mind, and experimental results showed the practicality of the approach. Finally, we extended the protocol to cope with collusions between the publisher and (a minority of) broken tokens, focusing on the prevention of the forge tampering action.

7.2.3 The Meta-Protocol Suite

The META-PROTOCOL suite generalizes the key findings of PROTOGEN by abstracting them from both the notion of equivalence classes and the k-ANONYMITY model, and dropping the constraint of a single step per phase. The honest-but-curious version of the protocol consequently consists in the following. During the **collection** phase,

each token sends a tuple consisting of (1) their records encrypted, (2) the algorithm-dependent data enabling the participation of the publisher in the **construction** phase, and (3) additional information for guaranteeing the correctness and security of the protocol, During the **construction** phase, the publisher forms the sanitization rules based on the information sent in the clear during the **collection** phase. Computing the sanitization rules data structure is the reason for centralizing data in centralized publishing algorithms: the sanitized release is produced based on these rules. Indeed, during the **sanitization** phase, each connecting token downloads a subset of tuples with their corresponding rules, and returns the corresponding sanitized records. The mapping between subsets of collected tuples and their corresponding subsets of sanitized records had to be precluded - we defined the `Unlinkability` safety property to this end. We then tackled the weakly-malicious attack model mainly by adapting the safety properties identified for PROTOGEN to this different context, and proposed corresponding proof-of-concept implementations. Experimental results assessed the practicality of the META-PROTOCOL instantiated as the $\alpha\beta$-ALGORITHM and enforcing the (d, γ)-PRIVACY model. The correctness and security of the resulting execution sequences were demonstrated against honest-but-curious and weakly-malicious publishers.

7.2.4 Practical Adequacy

Chapter 6 has surveyed the existing *electronic health record* approaches worldwide (*EHR* for short). It then proposed the implementation of an EHR system based on USB stick tokens in order to tackle the security and availability issues of existing EHR approaches. Finally, it discussed the practical implementation (i.e., setting and execution) of the PROTOGEN and META-PROTOCOL suite of protocols.

7.3 Perspectives

The work conducted in this thesis can be pursued in various directions. We identify below some challenging issues and outline possible lines of thought to tackle them. We start by issues that concern the internals of the META-PROTOCOL in the context considered in this thesis. Then, we enlarge the scope to other contexts related to privacy-preserving data publishing. Finally, we consider global treatments on a token-based architecture.

7.3.1 Concerning the Internals of the Meta-Protocol

Exploring the (Generality, Genericity) Space

Figure 7.1 qualitatively positions the Meta-Protocol with respect to the existing decentralized approaches handling centralized publishing models and algorithms. The reader will easily note the (almost) blank space described by the generality and genericity axes. Exploring this space by trading one or the other dimension will notably allow future works to tailor the implementations of the safety properties according to the underlying application.

For example, we could relax the **Genericity** objective to algorithms based on equivalence classes (e.g., generalization-based algorithms, bucketization-based algorithms). A dedicated implementation of the `Unlinkability`'s shuffle could benefit from this relaxation by shuffling each equivalence class internally only.

As another example, assume that the underlying application employs two types of tokens (e.g., doctors' and patients' tokens), and that one token type exhibits a low availability (e.g., patients' tokens) while the other type provides a high availability (e.g., doctors' tokens). The partitioning scheme could be tuned to produce two distinct partitionings: one with a large partition size for highly available tokens, and the other with a smaller partition size for lowly available tokens. The execution sequence would let each type of tokens work independently on its corresponding partitioning such that the partitions treated by the two token types do not overlap. How would the safety properties implementations (e.g., the shuffle) be adapted to still preclude tampering actions?

Broken Tokens

We initiated in Protogen the investigation of protection methods against a publisher colluding with a set of broken tokens. We plan to pursue these investigations in the context of the Meta-Protocol. The implementations of safety properties will have to cope with a clustered cryptographic material. This appears to be challenging. For example, a token participating in the shuffle currently downloads several buckets, and decrypts, shuffles, and re-encrypts the tuples before uploading them. However, with clustered cryptographic keys, a subset of downloaded tuples can have been encrypted based on a key different from the key of the connected token. The token will thus be unable to decrypt the buckets fully. If the token treats only the tuples originating from its own cluster, the publisher may infer the tuples' cluster based on the number of tuples returned. How can the shuffle be adapted to cope with these clustered tuples? A possible way could be based on introducing as many random bitstrings as the

number of tuples from other clusters. These false tuples, indistinguishable from true encrypted tuples, would thwart the previous inferences. Adapting the shuffling to this partial treatment of buckets and to the introduction of false tuples requires further investigations.

Designing Privacy Models and Algorithms for the Asymmetric Architecture

The ASYMMETRIC architecture introduces specific requirements (i.e., the functional and security requirements) concerning the algorithms that it is able to support. Investigating the design of privacy models and algorithms specifically designed to fit the ASYMMETRIC architecture is an open avenue for future work. Sampling-based schemes are especially interesting. Indeed, despite its simplicity, sampling has been shown to provide strong privacy guarantees (e.g., [Chaudhuri06] enforces DIFFERENTIAL PRIVACY based on sampling only). Moreover, recent investigations suggest a new privacy paradigm based on the *participation of records into the execution of the sanitization algorithm* [Kifer11]. Sampling seems a natural method for providing privacy guarantees in this paradigm. Note that this paradigm is different from the DIFFERENTIAL PRIVACY paradigm which focuses on the *participation of records into the sanitized release*. Finally, sampling naturally fits our highly decentralized token-based approach.

7.3.2 Concerning Other Privacy-Preserving Data Publishing Contexts

Data Types

This thesis focuses on tabular data. However, as highlighted in [Allard10a], tokens may store various data types such as, e.g., location data. Although this data may be representable in tabular format, sanitizing them requires specific sanitization models and algorithms that take into account the semantics of the underlying data (e.g., see [Giannotti08, Chow11, Terrovitis11] for privacy-preserving location data publishing). The information that must be disclosed voluntarily to the publisher in order to support a targeted data model may require to trade the genericity or generality dimension. For example, the sanitization of trajectories is often based on aggregating them [Chow11]. In our context, trajectories must not be disclosed to the publisher: how can they be aggregated?

Continuous Data Publishing

In this thesis, we considered publishing the dataset a single time. However, individuals evolve over time, and so do their data: records are inserted, deleted, updated. Continuously publishing an evolving dataset without disclosures is a challenging problem requiring its own models and algorithms (see, e.g., [Byun06, Xiao07, Bu08, Fung08, Byun09]). Continuous data publishing models usually require keeping a certain history of the data already released in order to compute the following release. How can this information be maintained and shared in our highly decentralized token-based approach? The `Invariance` safety property will probably have a role to play.

Interactive Privacy-Preserving Data Publishing

Rather than publishing a complete sanitized dataset, the interactive publishing approach proposes to publish answers to statistical queries posed over a private dataset. The control over the queries posed over the dataset give to interactive publishing algorithms an additional flexibility with respect to data sanitization, possibly resulting in better utility than the non-interactive approach [Rastogi07]. The current de facto interactive model is Differential Privacy; it is traditionally enforced through a controlled addition of noise into the results of aggregate queries.

Interestingly, this approach shares some similarities with the continuous publishing approach in that the noise added do the result of a given query notably depends on the queries previously answered [Dwork06b]. Maintaining and sharing an history is still a challenge here. Note that the dataset over which queries are posed may be formed in a way similar to non-interactive approaches: data may be collected from tokens and then be made invariant. The impact that the natural sampling of a token-based approach has on privacy guarantees could also be interesting to study in the context of Differential Privacy.

7.3.3 Concerning Global Function Evaluation with Tokens

How can the findings of Meta-Protocol benefit to the evaluation of other global functions over a dataset distributed over tokens? Except `Safe Rules`, the safety properties are not tied to the privacy-preserving data publishing approach: the `Invariance` property can fix any data structure prior to any function evaluation, the `Origin` property is a straightforward use of data integrity and authentication schemes, the `Execution Sequence Integrity` property can chain any execution sequence, and the `Identifier Unicity`, `Membership`, and `Mutual Exclusion` properties can check the absence of copies within any dataset. Moreover, even though the `Safe Rules` property

arises from the privacy-preserving data publishing context, the secure `Count` mechanism can be used to count anything out of a given dataset. Future works will identify global functions common to many database problems and adapt and extend the safety properties defined and implemented in this thesis.

Appendix A

Instanciation of $\alpha\beta$-Algorithm under Meta-Protocol

Contents

A.1 $\alpha\beta$-Algorithm$_{(hc)}$

Algorithm 4 describe the collection, construction, and sanitization phases of $\alpha\beta$-algorithm$_{(hc)}$. It contains calls to the `Sanitize` and `Unlinkability` procedures detailed after.

The `Sanitize` procedure, described in Algorithm 5, forms the obfuscated sanitized release partitition by partitition.

Finally, the `Unlinkability` procedure described in Algorithm 6 shuffles the obfuscated sanitized release.

A.2 $\alpha\beta$-Algorithm$_{(wm)}$

Algorithm 7 details the sanitization phase of $\alpha\beta$-algorithm$_{(wm)}$ from Step S1 to Step S3, and Algorithm 8 from Step S4 to Step S6. Peripheral functions are detailed in peripheral algorithms: the `Invariance` (Algorithm 9), `Mutual Exclusion` (Algorithm 10), secure sum (Algorithm 11), and `Unlinkability` (Algorithm 13) implementation techniques, and the `Sanitize` function (which consists in checking the `Origin`, `Membership`, and `Identifier Unicity` safety properties within a partition, and performing the sanitization action - Algorithm 12). These algorithms aim at favouring the clarity of the presentation, sometimes against the execution efficiency. Note that for the sake of clarity we focus on the enforcement of the safety properties during the **sanitization** phase, omitting the **collection** and **construction** phases (notably the technique for generating false tuples based on the intervals data structure) - they would only present a limited interest here.

Algorithm 4: $\alpha\beta$-algorithm$_{(hc)}$ - Global Execution Sequence

req. (publisher): The sampling constraint (i.e., the cardinality that the collected dataset must meet before switching to the construction phase), the privacy parameters (α, β, the dataset's cardinality $n_{\mathcal{D}_o}$, the rules's cardinality $n_{\mathcal{R}}$, the domain of all records), the branching factor f.

req. (tokens) : The cryptographic-material κ, some privacy parameters (α, β, the domain of all records), the branching factor f.

begin collection phase

  Each token that connects sends his record d in the form of a tuple $t \leftarrow (o, c, a)$, where $o \leftarrow \mathtt{E}_\kappa(d)$, $c \leftarrow \mathtt{M}^\sharp_\kappa(d)$, and $a \leftarrow$ "$\mathcal{D}_o$";

  The publisher forms its data structures: $\mathcal{D}_o \leftarrow \cup(t.o, t.a)$, $\mathcal{C} \leftarrow \cup(t.c)$;

  When the obfuscated dataset meets the required cardinality, the publisher switches to the construction phase;

end

begin construction phase

  Each token that connects sends several tuples (the actual number depends on the connection's duration) in the form: $t \leftarrow (o, c, a)$, where $t.o \leftarrow \mathtt{E}_\kappa(d)$, $t.c \leftarrow \mathtt{M}^\sharp_\kappa(d)$ and $t.a \leftarrow$ "$\mathcal{R}$", d being a record randomly sampled from the domain of all records;

  The publisher forms its data structures: $\mathcal{R} \leftarrow \cup(t.o, t.a)$ where $t.c \notin \mathcal{C}$, and $\mathcal{C} \leftarrow \cup(t.c)$;

end

begin sanitization phase

  The publisher forms the partitioned dataset $\mathcal{P}$ by performing the set union between the obfuscated dataset and the rules, and partitioning the resulting data structure: $\mathcal{P} \leftarrow \mathtt{P}(\mathcal{D}_o \cup \mathcal{R})$;

  forall $\mathcal{P}_i \in \mathcal{P}$ **do**

    Send $\mathcal{P}_i$ to a connecting token and add the returned obfucated partition to $\mathcal{V}_o$: $\mathcal{V}_o \leftarrow \mathcal{V}_o \cup$`Sanitize(`$\mathcal{P}_i$`)`;

  if *the number of partitions in* $\mathcal{V}_o$ *is not a multiple of a power of the branching factor* **then**

    The publisher inserts into $\mathcal{V}_o$ enough artificial tuples to meet the above condition;

  Tokens shuffle the obfuscated sanitized release and de-obfuscate it (ignoring artificial tuples): $\mathcal{V} \leftarrow$`Unlinkability(`$\mathcal{V}_o$, f`)`;

end

Algorithm 5: $\alpha\beta$-algorithm$_{(hc)}$ - `Sanitize`

req. (tokens): The cryptographic-material (κ), the privacy parameters (α and β).
input : The partition to sanitize ($\mathcal{P}_i$).
output : The obfuscated sanitized partition ($\mathcal{V}_{o,i}$).

Initialize the set of sanitized tuples: $T \leftarrow \varnothing$;
repeat
  Choose randomly a tuple $t \in \mathcal{P}_i$;
  if $t.a.l =$"$\mathcal{D}_o$" **then**
    With probability $1 - (\alpha + \beta)$: skip this tuple and go to the next element of the loop;
  Decrypt and re-encrypt the obfuscated part: $t_s \leftarrow \mathtt{E}_\kappa(\mathtt{E}_\kappa^{-1}(t.o))$;
  Add t_s to the set of sanitized tuples: $T \leftarrow T \cup t_s$;
until $\mathcal{P}_i = \varnothing$;
return T

Algorithm 6: $\alpha\beta$-algorithm$_{(hc)}$ - Performing `Unlinkability`

req. (tokens) : The cryptographic-material κ
input : The obfuscated sanitized release $\mathcal{V}_o$, the branching factor f
output : The sanitized release $\mathcal{V}$.

The publisher computes the shuffling circuit (including m) based on the number of partitions in $\mathcal{V}_o$ ($|\mathcal{V}_o|_p$) and the branching factor f (see Section 5.4). ;
repeat
  To each connecting token $\theta \in \Theta$, send f buckets ((or m during the last level) chosen according to the shuffling circuit;
  begin On θ
    Let T_{in} denote the set of encrypted tuples resulting from the union of the buckets downloaded, and T_{out} denote the tuples re-encrypted and shuffled;
    repeat
      Choose randomly an encrypted tuple $t_o \in T_{in}$;
      Delete it from T_{in}: $T_{in} \leftarrow T_{in} \backslash t_o$;
      Decrypt and re-encrypt it: $t_o \leftarrow \mathtt{E}_\kappa(\mathtt{E}_\kappa^{-1}(t_o))$;
      Add t_o to T_{out};
    until $T_{in} = \varnothing$;
    Send T_{out} to the publisher;
  end
  The publisher receives T_{out}, splits it into f buckets, and assigns to each bucket its position in the shuffling circuit (as explained in Section 5.4);
until *the shuffling circuit has been completely performed* ;
The publisher launches the decryption of the shuffled buckets and obtains $\mathcal{V}$;
return $\mathcal{V}$

Algorithm 7: $\alpha\beta$-algorithm$_{(wm)}$ - Sanitization Phase from Step S1 to Step S3

req. (publisher): The collected dataset ($\mathcal{D}_o$), the sanitization rules ($\mathcal{R}$), the set of labels for data structures to be made invariant (L), the number of tokens in the designated population (n_{pop}).

req. (tokens) : The cryptographic-material κ, the privacy parameters (α, β, the expected number of tuples in $\mathcal{D}_o$ ($n_{\mathcal{D}_o}$) and in $\mathcal{R}$ ($n_{\mathcal{R}}$)), the number of tokens in the designated population (n_{pop}).

The publisher forms the partitioned dataset $\mathcal{P}$ by performing the set union between the obfuscated dataset and the rules, and partitioning the resulting data structure: $\mathcal{P} \leftarrow \mathtt{P}(\mathcal{D}_o \cup \mathcal{R})$;

The publisher computes $\mathcal{P}$'s summary $\mathcal{S}^{\mathcal{P}}$ which maps each partition's TID-Set (expressed as a range) to its digest. Let $\mathcal{S}^{\mathcal{P}}.T$ denote the list of partitions's ranges (ordered), and $\mathcal{S}^{\mathcal{P}}(\tau)$ denote the digest to which the range $\tau \in \mathcal{S}^{\mathcal{P}}.T$ maps;

begin Step S1: `Invariance` of $\mathcal{P}$

  The publisher launches the `Invariance` on $\mathcal{S}^{\mathcal{P}}$ and gets the `Invariance` proof, i.e., the count increment representing the number of designated tokens having received $\mathcal{S}^{\mathcal{P}}$: $p_{inv} \leftarrow$`Invariance(`$L.\mathcal{P}$, $\mathcal{S}^{\mathcal{P}}$`)`;

  The publisher sends $\mathcal{S}^{\mathcal{P}}$ and p_{inv} to a connecting token; the token checks that p_{inv} is valid and that the observed count in p_{inv} is greater than the absolute majority of the designated population, counts the number of partitions in $\mathcal{S}^{\mathcal{P}}$, signs it, and sends the signature (denoted $p_{|\mathcal{P}|_p}$) to the publisher;

end

begin Step S2: `Mutual Exclusion` of $\mathcal{P}$

  The publisher asks a connecting token to check the `Mutual Exclusion` on $\mathcal{S}^{\mathcal{P}}$. It has to provide p_{inv} in order to prove that $\mathcal{S}^{\mathcal{P}}$ has been made invariant, and gets the `Mutual Exclusion` proof if the check succeeds: $p_{mutex} \leftarrow$`Mutual Exclusion(`$\mathcal{S}^{\mathcal{P}}$, p_{inv}, $L.\mathcal{P}$`)`;

end

begin Step S3: Process each partition in $\mathcal{P}$

  forall $\mathcal{P}_i \in \mathcal{P}$ **do**

    The publisher sends $\mathcal{P}_i$, $\mathcal{S}^{\mathcal{P}}$, and p_{mutex} to a connecting token, which performs the `Sanitize` function and returns $\mathcal{P}_i$ sanitized and obfuscated as well as the count increments of the partition: $(T, ci^{\mathcal{P}}_{\mathcal{D}_o}, ci^{\mathcal{P}}_{\mathcal{R}}, ci^{\mathcal{P}}_{drop}) \leftarrow$`Sanitize(`$\mathcal{P}_i$, $\mathcal{S}^{\mathcal{P}}$, p_{mutex}`)`;

    The publisher adds the obfuscated sanitized records to the obfuscated sanitized release: $\mathcal{V}_o \leftarrow \mathcal{V}_o \cup T$, and each count increment to its set of count increments: $counts^{\mathcal{P}}_{\mathcal{D}_o} \leftarrow counts^{\mathcal{P}}_{\mathcal{D}_o} \cup ci^{\mathcal{P}}_{\mathcal{D}_o}$, $counts^{\mathcal{P}}_{\mathcal{R}} \leftarrow counts^{\mathcal{P}}_{\mathcal{R}} \cup ci^{\mathcal{P}}_{\mathcal{R}}$, $counts^{\mathcal{P}}_{drop} \leftarrow counts^{\mathcal{P}}_{drop} \cup ci^{\mathcal{P}}_{drop}$;

  The publisher partitions $\mathcal{V}_o$: $\mathcal{V}_o \leftarrow \mathtt{P}(\mathcal{V}_o)$;

end

Algorithm 8: $\alpha\beta$-algorithm$_{(wm)}$ - Sanitization Phase from Step S4 to Step S6

req. (publisher): The partitionned obfuscated sanitized release ($\mathcal{V}_o$), the set of count increments related to $\mathcal{P}$ ($counts^{\mathcal{P}}_{\mathcal{D}_o}$, $counts^{\mathcal{P}}_{\mathcal{R}}$, and $counts^{\mathcal{P}}_{drop}$), the certified number of partitions in $\mathcal{P}$ ($p_{|\mathcal{P}|_p}$) the branching factor f, the set of labels for data structures to be made invariant (L).

req. (tokens) : The cryptographic-material (κ), the privacy parameters (α, β, the expected number of tuples in $\mathcal{D}_o$ ($n_{\mathcal{D}_o}$) and in $\mathcal{R}$ ($n_{\mathcal{R}}$)), the branching factor (f), the set of labels for data structures to be made invariant (L).

begin Step S4: Check `Safe Rules`

  The publisher launches the sum of the count increments for `Safe Rules`: $sum^{\mathcal{P}}_{\mathcal{D}_o} \leftarrow$`Sum(`$counts^{\mathcal{P}}_{\mathcal{D}_o}$`)` and $sum^{\mathcal{P}}_{\mathcal{R}} \leftarrow$`Sum(`$counts^{\mathcal{P}}_{\mathcal{R}}$`)` ;

  The publisher sends the sums and the certified number of partitions in $\mathcal{P}$ to a connecting token, which checks `Safe Rules` and returns a proof certifying that the rules have been checked: $p_r \leftarrow$`Safe Rules(`$sum^{\mathcal{P}}_{\mathcal{D}_o}$, $sum^{\mathcal{P}}_{\mathcal{R}}$, $p_{|\mathcal{P}|_p}$`)`;

end

begin Steps S5 and S6: `Unlinkability` of $\mathcal{V}$ and disclose $\mathcal{V}$

  if *the number of partitions in* $\mathcal{V}_o$ *is not a multiple of a power of the branching factor* **then**

    The publisher inserts into $\mathcal{V}_o$ enough artificial tuples to meet the above condition;

  The publisher launches the sum of the count increments for the tuples dropped: $sum^{\mathcal{P}}_{drop} \leftarrow$`Sum(`$counts^{\mathcal{P}}_{drop}$`)`;

  The publisher computes $\mathcal{V}_o$'s summary ($\mathcal{S}^{\mathcal{V}_o}$) similarly to $\mathcal{S}^{\mathcal{P}}$, based on which tokens assert the `Invariance` and `Mutual Exclusion` properties (similarly to their counterparts for $\mathcal{P}$). If these checks succeed, the publisher gets the corresponding proof p_{mutex};

  The publisher launches the check of the `Origin`, `Membership`, and `Identifier Unicity` properties on $\mathcal{V}_o$ (similarly to their counterparts for $\mathcal{P}$) and obtains the counts of "$\mathcal{D}_o$" ($sum^{\mathcal{V}_o}_{\mathcal{D}_o}$) and "$\mathcal{R}$" ($sum^{\mathcal{V}_o}_{\mathcal{R}}$) tuples in $\mathcal{V}_o$;

  The publisher obtains the certified number of partitions in $\mathcal{V}_o$ ($p_{|\mathcal{V}_o|_p}$) (similarly to its counterpart for $\mathcal{P}$) and the proof that $\mathcal{V}_o$'s counts are consistent with $\mathcal{P}$'s counts (p_{counts});

  The obfuscated sanitized release is shuffled and de-obfuscated: $\mathcal{V} \leftarrow$`Unlinkability(`$\mathcal{V}_o$, $|\mathcal{V}_o|_p$, $p_{|\mathcal{V}_o|_p}$, $\mathcal{S}^{\mathcal{V}_o}$, p_{counts}`)`;

end

Algorithm 9: $\alpha\beta$-algorithm$_{(wm)}$ - `Invariance`

req. (tokens): The cryptographic-material (κ), a boolean indicating if the token is part of the designated population or not (b_{pop}), the set of labels for data structures to be made invariant (L), a map (m) associating each label (l) to a boolean ($m(l)$) indicating whether the token already participated in the `Invariance` of the data structure corresponding to l.

input : A couple (l, DS) where DS denotes the data structure to be made invariant and l its label.

output : A proof that the couple (l, DS) is invariant.

Let *counts* denote the set of count increments related to the `Invariance` of (l, DS): $counts \leftarrow \varnothing$ and n_{counts} denote its cardinality;
repeat
    Let $\theta \in \Theta$ be a connecting token;
    if $\theta.b_{pop} = 1$ *and* $\theta.m(l) = 0$ **then**
        The publisher sends to θ the couple (l, DS);
        begin On θ
            Hash the couple received: $h \leftarrow \mathtt{H}(l||DS)$;
            Compute a random identifier r;
            Produce a new count increment: $c \leftarrow$ (1,"`Invariance` of "$||h, r : r, \mathtt{M}_{\kappa}$(1,"`Invariance` of "$||h||r : r$));
            And send c to the publisher;
        end
        The publisher adds c to the set of count increments: $counts \leftarrow counts \cup c$;
until $n_{counts} \geq 0.5 \cdot n_{pop} + 1$;
The publisher gets the `Invariance` proof, which is the sum of the count increments in *counts*: $sum \leftarrow$`Sum(`*counts*`)`;
return *sum*;

Algorithm 10: $\alpha\beta$-algorithm$_{(wm)}$ - `Mutual Exclusion` (on a Token)

req. (tokens): The cryptographic-material (κ), the size of the designated population (n_{pop}).

input : A summary ($\mathcal{S}$), the sum of the count increments related to the `Invariance` of $\mathcal{S}$ (p_{inv}), the label of the dataset (l).

output : The proof of `Mutual Exclusion` for $\mathcal{S}$ if `Mutual Exclusion` is asserted.

1 Check that the sum's signature is valid: $sum.\sigma = \mathtt{M}_{\kappa}(sum.o||sum.p||sum.r)$;

2 Hash the couple received: $h \leftarrow \mathtt{H}(l||\mathcal{S})$;

3 Check that the sum's purpose is valid: $sum.p =$ "`Invariance` of "$||h$;

4 Check that the absolute majority of the designated population has received the summary: $sum.c \geq 0.5 \cdot n_{pop} + 1$;

5 Check that the ranges of the summary respect the `Mutual Exclusion` safety property by iterating over the list of ranges $\mathcal{S}.T$ in ascending order and comparing each range τ_l with its right neighbour τ_r: assert that $\tau_l \cap \tau_r = \varnothing$ and $\tau_l.max < \tau_r.min$.

6 Compute the `Mutual Exclusion` proof: $p_{mutex} \leftarrow \mathtt{M}_{\kappa}($ "`Mutual Exclusion` of "$||h)$;

7 **return** p_{mutex}

Algorithm 11: $\alpha\beta$-algorithm$_{(wm)}$ - `Sum` (on a Token)

req. (tokens): The cryptographic-material (κ).
input : A set of count increments *counts*.
output : A count increment representing the sum of *counts*.

repeat
  The publisher orders the set of count increments by identifier range and partitions it;
  $tmp \leftarrow counts$;
  $counts \leftarrow \varnothing$;
  forall *partition* $pc \subset tmp$ **do**
    The publisher sends pc to a connecting token θ;
    begin On θ

      `// Check the count increments's integrity`
      Check that the ranges of count increments do not overlap by iterating over them in ascending order and comparing each range $c_l.r$ with its right neighbour $c_r.r$: assert that $c_l.r \cap c_r.r = \varnothing$ and $c_l.r.max < c_r.r.min$;
      Check the validity of the signature of all count increments: $\forall c \in pc,\ c.\sigma = \mathtt{M}_\kappa(c.o, c.p, c.r, c.n)$;
      Check that all count increments share the same purpose: $\forall (c_i, c_j) \in pc \times pc,\ c_i.p = c_j.p$;

      `// Form the count increment representing the sum (`c_s`)`
      Compute the sum of the counts observed:$c_s.o \leftarrow \sum_{\forall c \in pc} c.o$;
      The purpose of c_s is the unique purpose of the count increments: $c_s.p \leftarrow c_i.p$ where $c_i \in pc$. The range of c_s is formed by the minimal and maximal identifiers covered by the set of count increments. Let denote them respectively r_{min} and r_{max}: $c_s.r \leftarrow r_{min} : r_{max}$;
      if *count increments have a field "number"* **then**
        The number of partitions covered by the sum is: $c_s.n \leftarrow \sum_{\forall c \in pc} c.n$;
        Sign c_s: $c_s.\sigma \leftarrow \mathtt{M}_\kappa(c_s.o||c_s.p||c_s.r||c_s.n)$;
      else
        Sign c_s: $c_s.\sigma \leftarrow \mathtt{M}_\kappa(c_s.o||c_s.p||c_s.r)$;
      Send c_s to the publisher;
    end
    The publisher adds c_s to *counts*: $counts \leftarrow counts \cup c_s$;
until *counts contains a single count increment* ;
return *counts*

Algorithm 12: $\alpha\beta$-algorithm$_{(wm)}$ - `Sanitize` (on a Token)

req. (tokens): The cryptographic-material (κ), the privacy parameters (α and β), the set of labels for data structures to be made invariant (L).

input : The partition to sanitize ($\mathcal{P}_i$), $\mathcal{P}$'s summary ($\mathcal{S}^{\mathcal{P}}$), the `Mutual Exclusion` proof of $\mathcal{S}^{\mathcal{P}}$ (p_{mutex}).

output : A set of encrypted sanitized tuples.

Check p_{mutex}: $p_{mutex} = \mathtt{M}_\kappa(\text{“}\texttt{Mutual Exclusion}\text{ of ”}||\mathtt{H}(L.\mathcal{P}||\mathcal{S}^{\mathcal{P}}))$;

`// Check the consistency between the summary and the partition`

$\mathcal{P}_i$'s range exists in the summary: $\exists\tau \in \mathcal{S}^{\mathcal{P}}.T$ s.t. $\tau = \mathcal{P}_i^{\tau}$;

The summary's digest is equal to the hash of the $\mathcal{P}_i$'s tuple: $\mathcal{S}^{\mathcal{P}}(\mathcal{P}_i^{\tau}) = \mathtt{H}(\mathcal{P}_i)$;

`// Check the safety properties`

Initialize the set of tuple identifiers in the partition: $ids \leftarrow \varnothing$;

forall *tuple* $t \leftarrow (o, a, \sigma) \in \mathcal{P}_i$ **do**

  Check the `Origin` property: $t.\sigma = \mathtt{M}_\kappa(t.o||t.a)$;

  Check the `Identifier Unicity` property: $t.a.\tau \notin ids$;

  Check the `Membership` property: $t.a.\tau \in \mathcal{P}_i^{\tau}$;

  $ids \leftarrow ids \cup t.a.\tau$;

`// Sanitize tuples`

Initialize the set of sanitized tuples: $T \leftarrow \varnothing$;

Initialize the counters of the $\mathcal{R}$ and $\mathcal{D}_o$ tuples, and of the dropped tuples: $c_{\mathcal{R}} \leftarrow 0$, $c_{\mathcal{D}_o} \leftarrow 0$, $c_{drop} \leftarrow 0$;

repeat

  Choose randomly a tuple $t \in \mathcal{P}_i$, remove it from $\mathcal{P}_i$: $\mathcal{P}_i \leftarrow \mathcal{P}_i \backslash t$;

  if $t.a.l =$“$\mathcal{D}_o$” **then**

    $c_{\mathcal{D}_o}++$;

    With probability $1-(\alpha+\beta)$: drop the tuple, increment the counter of dropped tuples $c_{drop}++$, and go to the next element of the loop;

  else

    $c_{\mathcal{R}}++$;

  Decrypt the obfuscated part: $d \leftarrow \mathtt{E}_\kappa^{-1}(t.o)$;

  Form the sanitized encrypted tuple $t_s \leftarrow \mathtt{E}_\kappa(d, t.\tau, \mathtt{M}_\kappa(d, t.\tau))$;

  Add t_s to the set of sanitized tuples: $T \leftarrow T \cup t_s$;

until $\mathcal{P}_i = \varnothing$;

Form the count increment of $\mathcal{D}_o$ tuples: $ci_{\mathcal{D}_o} \leftarrow (c_{\mathcal{D}_o}, p, r, n)$ where $p \leftarrow$“$\mathcal{D}_o$”, $r \leftarrow \mathcal{P}_i.pos : \mathcal{P}_i.pos$, and $n \leftarrow 1$;

The count increment of $\mathcal{R}$ tuples, $ci_{\mathcal{R}}$, is similar;

Form the count increment for dropped tuples: $ci_{drop} \leftarrow (c_{drop}, p, r, n)$ where $p \leftarrow$“drop”, $r \leftarrow \mathcal{P}_i.pos : \mathcal{P}_i.pos$, and $n \leftarrow 1$;

return $(T, ci_{\mathcal{D}_o}, ci_{\mathcal{R}}, ci_{drop})$

Algorithm 13: $\alpha\beta$-algorithm$_{(wm)}$ - `Unlinkability`

req. (tokens) : The cryptographic-material κ, the branching factor f.
req. (publisher): The branching factor f.
input : The obfuscated sanitized release ($\mathcal{V}_o$), its summary ($S^{\mathcal{V}_o}$), the proof that $S^{\mathcal{V}_o}$ respects `Mutual Exclusion` (p_{mutex}),the certified number of partitions in $\mathcal{V}_o$ ($|\mathcal{V}_o|_p$ and $p_{|\mathcal{V}_o|_p}$), the proof that $\mathcal{V}_o$'s counts are consistent with $\mathcal{P}$'s counts (p_{counts}).
output : The sanitized release $\mathcal{V}$.

The publisher computes the shuffling circuit based on $|\mathcal{V}_o|_p$ and f (Section 5.4);
repeat
  To each connecting token $\theta \in \Theta$, send $|\mathcal{V}_o|_p$ with its signature $p_{|\mathcal{V}_o|_p}$, and f buckets (except during the first level (send one partition) and during the last level (send m buckets)) chosen according to the shuffling circuit;
  During the first level, tokens check the validity of p_{mutex}, the consistency between the partition received and the summary $\mathcal{S}^{\mathcal{V}_o}$, the validity of p_{counts};
  Tuples contained in bucket $b \in B$ are denoted $b.T$. Moreover, in order to guarantee the `Execution Sequence Integrity` of the shuffling circuit, each bucket b is accompanied with its position, denoted $b.pos$, and a signature of its tuples and position, denoted $b.\sigma$;
  begin On θ
    Check the validity of $|\mathcal{V}_o|_p$: $p_{|\mathcal{V}_o|_p} = \mathtt{M}_\kappa(|\mathcal{V}_o|_p)$;
    Let B_{in} denote the set of buckets downloaded;
    Let $P \leftarrow \varnothing$;
    forall *bucket* $b \in B_{in}$ **do**
      Check the validity of b's signature: $b.\sigma = \mathtt{M}_\kappa(\mathtt{H}(b.T)||b.pos)$;
      $P \leftarrow b.pos$;
    According to the shuffling circuit, check that the positions in P must indeed be shuffled together (see Section 5.4);
    Let T_{in} denote the set of encrypted tuples in the buckets downloaded, and T_{out} denote the tuples re-encrypted and shuffled;
    repeat
      Choose randomly an encrypted tuple $t_o \in T_{in}$;
      Delete it from T_{in}: $T_{in} \leftarrow T_{in} \backslash t_o$;
      Decrypt and re-encrypt it: $t_o \leftarrow \mathtt{E}_\kappa(\mathtt{E}_\kappa^{-1}(t_o))$;
      Add t_o to T_{out};
    until $T_{in} = \varnothing$;
    Split T_{out} in a set B_{out} of f buckets (or m);
    foreach *bucket* $b \in B_{out}$ **do**
      Compute its position $b.pos$ (as explained in Section 5.4);
      Compute its signature: $b.\sigma \leftarrow \mathtt{M}_\kappa(\mathtt{H}(b.T)||b.pos)$;
    Send B_{out} (ordered by position) to the publisher;
  end
  The publisher receives B_{out}, and assigns to each bucket its position in the shuffling circuit (as explained in Section 5.4);
until *the shuffling circuit has been completely performed* ;
The publisher launches the decryption of the shuffled buckets and obtains $\mathcal{V}$ (tokens continue checking the `Execution Sequence Integrity`);
return $\mathcal{V}$

Bibliography

[Adam89] Nabil R. Adam & John C. Worthmann. *Security-control methods for statistical databases: a comparative study.* ACM Comput. Surv., vol. 21, pages 515–556, ACM, New York, NY, USA, December 1989.

Cited on Page(s): 15.

[Aggarwal06] Gagan Aggarwal, Tomás Feder, Krishnaram Kenthapadi, Samir Khuller, Rina Panigrahy, Dilys Thomas & An Zhu. *Achieving anonymity via clustering.* In Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '06, pages 153–162, New York, NY, USA, 2006. ACM.

Cited on Page(s): 25.

[Aggarwal08] Charu C. Aggarwal & Philip S. Yu. Privacy-preserving data mining: Models and algorithms. Springer Publishing Company, Incorporated, 1 edition, 2008.

Cited on Page(s): 14, 15.

[Agrawal02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant & Yirong Xu. *Hippocratic databases.* In Proceedings of the 28th international conference on Very Large Data Bases, VLDB '02, pages 143–154. VLDB Endowment, 2002.

Cited on Page(s): 153.

[Agrawal04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant & Yirong Xu. *Order preserving encryption for numeric data.* In Proceedings of the 2004 ACM SIGMOD international conference on Management of data, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.

Cited on Page(s): 94, 100.

[Agrawal05a] Rakesh Agrawal, Ramakrishnan Srikant & Dilys Thomas. *Privacy preserving OLAP*. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, SIGMOD '05, pages 251–262, New York, NY, USA, 2005. ACM.

Cited on Page(s): 30.

[Agrawal05b] Shipra Agrawal & Jayant 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.

Cited on Page(s): 30, 31.

[Agrawal09] Shipra Agrawal, Jayant R. Haritsa & B. Aditya Prakash. *FRAPP: a framework for high-accuracy privacy-preserving mining*. Data Min. Knowl. Discov., vol. 18, no. 1, pages 101–139, Kluwer Academic publishers, Hingham, MA, USA, February 2009.

Cited on Page(s): 6, 30, 31.

[Allard09] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Philippe Pucheral & Romuald Thion. *Seamless access to healthcare folders with strong privacy guarantees*. International Journal of Healthcare Delivery Reform Initiatives, vol. 1, no. 4, pages 82–107, IGI Global, 2009.

Cited on Page(s): 7, 131.

[Allard10a] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Yanli Guo, Lionel Le Folgoc, Benjamin Nguyen, Philippe Pucheral, Indrajit Ray, Indrakshi Ray & Shaoyi Yin. *Secure personal data servers: a vision paper*. Proc. VLDB Endow., vol. 3, pages 25–35, VLDB Endowment, September 2010.

Cited on Page(s): 6, 153, 159.

[Allard10b] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Philippe Pucheral & Romuald Thion. *Concilier Ubiquité et Sécurité des Données Médicales*. In Cahiers du CRID, editeur, Les technologies de l'information au service des droits : opportunités, défis, limites, volume 32, pages 173–219. Editions Bruylant, 2010. Chapitre IX.

Cited on Page(s): 7, 131.

[Allard10c] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Philippe Pucheral & Romuald Thion. *Trustworthiness of Pervasive Healthcare Folders*. In Antonio Coronato & Giuseppe De Pietro, editeurs, Pervasive and Smart Technologies for Healthcare, pages 172–196. IGI Global, 2010. Chapter IX.

Cited on Page(s): 7, 131.

[Allard11a] Tristan Allard, Benjamin Nguyen & Philippe 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. Best Paper Award.

Cited on Page(s): 7, 60.

[Allard11b] Tristan Allard, Benjamin Nguyen & Philippe 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.

Cited on Page(s): 7, 60.

[Allard11c] Tristan Allard, Benjamin Nguyen & Philippe Pucheral. *Towards a Safe Realization of Privacy-Preserving Data Publishing Mechanisms*. In Ph.D. Colloquium of the 12th International Conference on Mobile Data Management, 2011.

Cited on Page(s): 7, 60.

[Alliance03] Smart Card Alliance. *HIPAA Compliance and Smart Cards (Report No. ID-03004)*. Smart Card Alliance, 2003.

Cited on Page(s): 137.

[Alliance06] Smart Card Alliance. *Smart Card Applications in the U.S. Healthcare Industry (White Paper No. HC-06001)*. Smart Card Alliance Healthcare Council, 2006.

Cited on Page(s): 137.

[Anciaux01] Nicolas Anciaux, Christophe Bobineau, Luc Bouganim, Philippe Pucheral & Patrick Valduriez. *PicoDBMS: Validation and Experience*. In Proceedings of the 27th International Conference on

Very Large Data Bases, VLDB '01, pages 709–710, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

Cited on Page(s): 140.

[Anciaux06] Nicolas Anciaux, Luc Bouganim & Philippe Pucheral. *Data Confidentiality: to which extent cryptography and secured hardware can help.* Annales des Télécommunications, vol. 61, no. 3-4, pages 267–283, Lavoisier, 2006.

Cited on Page(s): 142.

[Anciaux07a] Nicolas Anciaux, Mehdi Benzine, Luc Bouganim, Philippe Pucheral & Dennis Shasha. *GhostDB: querying visible and hidden data without leaks.* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD '07, pages 677–688, New York, NY, USA, 2007. ACM.

Cited on Page(s): 141.

[Anciaux07b] Nicolas Anciaux, Luc Bouganim & Philippe Pucheral. *Future Trends in Secure Chip Data Management.* IEEE Data Engineering Bulletin, vol. 30, no. 3, pages 49–57, IEEE Computer Society, 2007.

Cited on Page(s): 140.

[Anciaux10] Nicolas Anciaux, Luc Bouganim, Yanli Guo, Philippe Pucheral, Jean-Jacques Vandewalle & Shaoyi 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.

Cited on Page(s): 79, 127.

[Ards98] Sheila Ards, Chanjin Chung & Jr. Myers Samuel L. *The effects of sample selection bias on racial differences in child abuse reporting.* Child Abuse & Neglect: The International Journal, vol. 22, pages 103–115, Pergamon Press, February 1998.

Cited on Page(s): 149.

[Arrington06] Michael Arrington. *AOL Proudly Releases Massive Amounts of Private Data.* TechCrunch, 6th of August 2006.

Cited on Page(s): 4.

[Ashrafi09a] Mafruz Zaman Ashrafi & See Kiong Ng. *Collusion-resistant anonymous data collection method.* In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '09, pages 69–78, New York, NY, USA, 2009. ACM.

Cited on Page(s): 46.

[Ashrafi09b] Mafruz Zaman Ashrafi & See Kiong Ng. *Efficient and Anonymous Online Data Collection.* In Proceedings of the 14th International Conference on Database Systems for Advanced Applications, DASFAA '09, pages 471–485, Berlin, Heidelberg, 2009. Springer-Verlag.

Cited on Page(s): 46.

[Barbaro06] Michael Barbaro & Tom Jr. Zeller. *A Face Is Exposed for AOL Searcher No. 4417749.* The New York Times, 9th of August 2006.

Cited on Page(s): 4.

[Barnett94] Vic Barnett & Toby Lewis. Outliers in statistical data. Wiley, 3rd edition, 1994.

Cited on Page(s): 85, 86.

[Bayardo05] Roberto J. Bayardo & Rakesh Agrawal. *Data Privacy through Optimal k-Anonymization.* In Proceedings of the 21st International Conference on Data Engineering, ICDE '05, pages 217–228, Washington, DC, USA, 2005. IEEE Computer Society.

Cited on Page(s): 25.

[Bertino05] Elisa Bertino, Igor Fovino & Loredana Provenza. *A Framework for Evaluating Privacy Preserving Data Mining Algorithms[*].* Data Mining and Knowledge Discovery, vol. 11, pages 121–154, Springer Netherlands, 2005. 10.1007/s10618-005-0006-6.

Cited on Page(s): 22.

[Bertino08] Elisa Bertino, Dan Lin & Wei Jiang. *A Survey of Quantification of Privacy Preserving Data Mining Algorithms.* In Charu C. Aggarwal, Philip S. Yu & Ahmed K. Elmagarmid, editeurs, Privacy-Preserving Data Mining, volume 34 of *The Kluwer International*

Series on Advances in Database Systems, pages 183–205. Springer US, 2008.

Cited on Page(s): 22.

[Bloom70] Burton H. Bloom. *Space/time trade-offs in hash coding with allowable errors.* Commun. ACM, vol. 13, pages 422–426, ACM, New York, NY, USA, July 1970.

Cited on Page(s): 141.

[Boldyreva09] Alexandra Boldyreva, Nathan Chenette, Younho Lee & Adam O'Neill. *Order-Preserving Symmetric Encryption.* In Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques, EUROCRYPT '09, pages 224–241, Berlin, Heidelberg, 2009. Springer-Verlag.

Cited on Page(s): 94, 100.

[Bos09] Lodewijk Bos. *Dutch nationwide EHR postponed. Are they in good company?* International Council on Medical and Care Compunetics, 23rd of January 2009.

Cited on Page(s): 5, 133, 138.

[Brickell06] Justin Brickell & Vitaly Shmatikov. *Efficient anonymity-preserving data collection.* In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '06, pages 76–85, New York, NY, USA, 2006. ACM.

Cited on Page(s): 46.

[Brickell08] Justin Brickell & Vitaly Shmatikov. *The cost of privacy: destruction of data-mining utility in anonymized data publishing.* In Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, pages 70–78, New York, NY, USA, 2008. ACM.

Cited on Page(s): 28.

[Brown03] Steven H. Brown, Michael J. Lincoln, Peter J. Groen & Robert M. Kolodner. *VistA - U.S. Department of Veterans Affairs national-scale HIS.* International Journal of Medical Informatics, vol. 69, no. 2-3, pages 135–156, 2003.

Cited on Page(s): 135.

[Bu08] Yingyi Bu, Ada Wai Chee Fu, Raymond Chi Wing Wong, Lei Chen & Jiuyong Li. *Privacy preserving serial data publishing by role composition.* Proc. VLDB Endow., vol. 1, pages 845–856, VLDB Endowment, August 2008.

Cited on Page(s): 150, 160.

[Byun06] Ji-Won Byun, Yonglak Sohn, Elisa Bertino & Ninghui Li. *Secure anonymization for incremental datasets.* In VLDB Workshop on Secure Data Management, SDM '06, pages 48–63, 2006.

Cited on Page(s): 150, 160.

[Byun09] Ji-Won Byun, Tiancheng Li, Elisa Bertino, Ninghui Li & Yonglak Sohn. *Privacy-preserving incremental data dissemination.* Journal of Computer Security, vol. 17, pages 43–68, IOS Press, Amsterdam, The Netherlands, The Netherlands, January 2009.

Cited on Page(s): 150, 160.

[Cao11] Jianneng Cao, Panagiotis Karras, Panos Kalnis & Kian-Lee Tan. *SABRE: a Sensitive Attribute Bucketization and REdistribution framework for t-closeness.* The VLDB Journal, vol. 20, pages 59–81, Springer-Verlag New York, Inc., Secaucus, NJ, USA, February 2011.

Cited on Page(s): 27.

[Carrasco99] Lionel C. Carrasco. *RDBMS's for Java Cards? What a Senseless Idea! (White Paper).* ISOL Corp., 1999.

Cited on Page(s): 140.

[Chaudhuri06] Kamalika Chaudhuri & Nina Mishra. *When Random Sampling Preserves Privacy.* In CRYPTO, pages 198–213, 2006.

Cited on Page(s): 159.

[Chen07] Bee-Chung Chen, Kristen LeFevre & Raghu Ramakrishnan. *Privacy skyline: privacy with multidimensional adversarial knowledge.* In Proceedings of the 33rd international conference on Very large data bases, VLDB '07, pages 770–781. VLDB Endowment, 2007.

Cited on Page(s): 17, 19, 105.

[Chen09a] Bee-Chung Chen, Daniel Kifer, Kristen LeFevre & Ashwin Machanavajjhala. *Privacy-Preserving Data Publishing*. Found. Trends in Databases, vol. 2, no. 1-2, pages 1–167, Now publishers Inc., Hanover, MA, USA, January 2009.

Cited on Page(s): 4, 5, 15, 19, 22, 31.

[Chen09b] Bee-Chung Chen, Kristen Lefevre & Raghu Ramakrishnan. *Adversarial-knowledge dimensions in data privacy*. The VLDB Journal, vol. 18, pages 429–467, Springer-Verlag New York, Inc., Secaucus, NJ, USA, April 2009.

Cited on Page(s): 19.

[Chow11] Chi-Yin Chow & Mohamed F. Mokbel. *Trajectory privacy in location-based services and data publication*. SIGKDD Explor. Newsl., vol. 13, pages 19–29, ACM, New York, NY, USA, August 2011.

Cited on Page(s): 159.

[Clifton02] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin & Michael Y. Zhu. *Tools for privacy preserving distributed data mining*. SIGKDD Explor. Newsl., vol. 4, pages 28–34, ACM, New York, NY, USA, December 2002.

Cited on Page(s): 45.

[Cormode10] Graham Cormode, Divesh Srivastava, Ninghui Li & Tiancheng Li. *Minimizing minimality and maximizing utility: analyzing method-based attacks on anonymized data*. Proc. VLDB Endow., vol. 3, pages 1045–1056, VLDB Endowment, September 2010.

Cited on Page(s): 29.

[Cormode11] Graham 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.

Cited on Page(s): 29.

[Cortes08] Corinna Cortes, Mehryar Mohri, Michael Riley & Afshin Rostamizadeh. *Sample Selection Bias Correction Theory*. In ALT, pages 38–53, 2008.

Cited on Page(s): 149.

[Dahl07] Mads Ronald Dahl. *Status and perspective of personal health informatics in Denmark*. Denmark: University of Aarhus, Section for Health Informatics, Institute of Public Health, 2007.

Cited on Page(s): 136.

[Dai09] Chenyun Dai, Gabriel Ghinita, Elisa Bertino, Ji-Won Byun & Ninghui Li. *TIAMAT: a tool for interactive analysis of microdata anonymization techniques*. Proc. VLDB Endow., vol. 2, pages 1618–1621, VLDB Endowment, August 2009.

Cited on Page(s): 29.

[Dalenius77] Tore Dalenius. *Towards a methodology for statistical disclosure control*. Statistik Tidskrift, vol. 15, no. 5, pages 429–444, 1977.

Cited on Page(s): 17.

[Dierks08] Tim Dierks & Eric Rescola. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246, Internet Engineering Task Force, 2008.

Cited on Page(s): 138, 143.

[Ding11] Bolin Ding, Marianne Winslett, Jiawei Han & Zhenhui Li. *Differentially private data cubes: optimizing noise sources and consistency*. In Proceedings of the 2011 international conference on Management of data, SIGMOD '11, pages 217–228, New York, NY, USA, 2011. ACM.

Cited on Page(s): 21.

[Door08] Jean-Pierre Door. *Le dossier m'edical personnel (Information Rep. No. 659)*. France: Assembl'ee Nationale, 2008.

Cited on Page(s): 133.

[Dwork06a] Cynthia Dwork. *Differential Privacy*. In Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, ICALP '06, pages 1–12, 2006.

Cited on Page(s): 17, 21.

[Dwork06b] Cynthia Dwork, Frank McSherry, Kobbi Nissim & Adam Smith. *Calibrating Noise to Sensitivity in Private Data Analysis*. In

Proceedings of the 3rd Theory of Cryptography Conference, TCC '06, pages 265–284, 2006.

Cited on Page(s): 21, 160.

[Dwork09] Cynthia Dwork & Jing Lei. *Differential privacy and robust statistics.* In Proceedings of the 41st annual ACM symposium on Theory of computing, STOC '09, pages 371–380, New York, NY, USA, 2009. ACM.

Cited on Page(s): 21.

[Dwork10] Cynthia Dwork. *Differential privacy in new settings.* In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10, pages 174–183, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

Cited on Page(s): 21.

[Dwork11] Cynthia Dwork. *A firm foundation for private data analysis.* Commun. ACM, vol. 54, pages 86–95, ACM, New York, NY, USA, January 2011.

Cited on Page(s): 21.

[Dworkin01] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques.* National Institute of Standards and Technologies (NIST), Special Publication 800-38A, 2001.

Cited on Page(s): 34, 35, 36.

[Dworkin05] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.* National Institute of Standards and Technology (NIST), Special Publication 800-38B, 2005.

Cited on Page(s): 40.

[Dworkin07a] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.* National Institute of Standards and Technology (NIST), Special Publication 800-38D, 2007.

Cited on Page(s): 35.

[Dworkin07b] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.* National Institute of Standards and Technologies (NIST), Special Publication 800-38C, 2007.

Cited on Page(s): 34, 35, 36.

[Dworkin10] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode.* National Institute of Standards and Technology (NIST), Addendum to NIST Special Publication 800-38A, 2010.

Cited on Page(s): 35.

[EPC95] EPC. *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.* European Parliament and Council, 1995.

Cited on Page(s): 3.

[Eurosmart08] Eurosmart. *Smart USB Token (White Paper).* Eurosmart, 2008.

Cited on Page(s): 139.

[Evfimievski03] Alexandre Evfimievski, Johannes Gehrke & Ramakrishnan Srikant. *Limiting privacy breaches in privacy preserving data mining.* In Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '03, pages 211–222, New York, NY, USA, 2003. ACM.

Cited on Page(s): 17, 30, 31.

[Fadem08] Barbara Fadem. Behavioral Science. Lippincott Williams & Wilkins, 5th edition, 2008.

Cited on Page(s): 149.

[FCSM05] FCSM. *Report on Statistical Disclosure Limitation Methodology.* Statistical Policy Working Paper 22 (Second version, 2005), Federal Committee on Statistical Methodology (FCSM), 2005.

Cited on Page(s): 62.

[Fischlin11] Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider & Ivan 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.

Cited on Page(s): 46.

[Fung05] Benjamin C. M. Fung, Ke Wang & Philip S. Yu. *Top-Down Specialization for Information and Privacy Preservation.* In Proceedings of the 21st International Conference on Data Engineering, ICDE '05, pages 205–216, Washington, DC, USA, 2005. IEEE Computer Society.

Cited on Page(s): 25, 45.

[Fung08] Benjamin C. M. Fung, Ke Wang, Ada Wai-Chee Fu & Jian Pei. *Anonymity for continuous data publishing.* In Proceedings of the 11th international conference on Extending database technology: Advances in database technology, EDBT '08, pages 264–275, New York, NY, USA, 2008. ACM.

Cited on Page(s): 150, 160.

[Fung10] Benjamin C. M. Fung, Ke Wang, Rui Chen & Philip S. Yu. *Privacy-preserving data publishing: A survey of recent developments.* ACM Comput. Surv., vol. 42, pages 14:1–14:53, ACM, New York, NY, USA, June 2010.

Cited on Page(s): 4, 22.

[Gantz11] John F. Gantz & David Reinsel. *Extracting Value From Chaos (White Paper).* IDC, 2011.

Cited on Page(s): 3.

[Ghinita07] Gabriel Ghinita, Panagiotis Karras, Panos Kalnis & Nikos 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.

Cited on Page(s): 25.

[Ghinita09] Gabriel Ghinita, Panagiotis Karras, Panos Kalnis & Nikos Mamoulis. *A framework for efficient data anonymization under privacy and accuracy constraints.* ACM Trans. Database Syst., vol. 34, pages 9:1–9:47, ACM, New York, NY, USA, July 2009.

Cited on Page(s): 25.

[Giannotti08] Fosca Giannotti & Dino Pedreschi. Mobility, data mining and privacy - geographic knowledge discovery. Mobility, Data Mining and Privacy. Springer, 2008.

Cited on Page(s): 159.

[Goldreich87] O. Goldreich, S. Micali & 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.

Cited on Page(s): 41.

[Goldreich01] Oded Goldreich. Foundations of cryptography: Basic tools. Cambridge University Press, New York, NY, USA, 2001.

Cited on Page(s): 32.

[Goldreich04] Oded Goldreich. Foundations of cryptography: Volume 2, basic applications. Cambridge University Press, New York, NY, USA, 1st edition, 2004.

Cited on Page(s): 32, 33, 34.

[Goldreich05] Oded Goldreich. *Foundations of cryptography: a primer.* Found. Trends in Theoretical Computer Science, vol. 1, no. 1, pages 1–116, Now publishers Inc., Hanover, MA, USA, April 2005.

Cited on Page(s): 32, 56.

[Golle06] Philippe Golle. *Revisiting the uniqueness of simple demographics in the US population.* In Proceedings of the 5th ACM workshop on Privacy in electronic society, WPES '06, pages 77–80, New York, NY, USA, 2006. ACM.

Cited on Page(s): 4.

[Google11] Google. *An update on Google Health and Google PowerMeter.* The Official Google Blog, 24th of June 2011.

Cited on Page(s): 135.

[Gordon06] Lawrence A. Gordon, Martin P. Loeb, William Lucyshin & Robert Richardson. *2006 CSI/FBI Computer Crime and Security Survey.* Computer Security Institute, 2006.

Cited on Page(s): 133, 138.

[Goyal10a] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody & Amit Sahai. *Interactive Locking, Zero-Knowledge PCPs, and Unconditional Cryptography.* In Tal Rabin, editeur, Advances in Cryptology

- CRYPTO 2010, volume 6223 of *Lecture Notes in Computer Science*, pages 173–190. Springer Berlin / Heidelberg, 2010.

Cited on Page(s): 46.

[Goyal10b] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan & Akshay Wadia. *Founding Cryptography on Tamper-Proof Hardware Tokens.* In Daniele Micciancio, editeur, Theory of Cryptography, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer Berlin / Heidelberg, 2010.

Cited on Page(s): 46.

[Gratzel08] Philipp Gratzel. *German doctors say no to centrally stored patient records.* eHealth Insider, 21st of January 2008.

Cited on Page(s): 138.

[Greengard08] Samuel Greengard. *Privacy matters.* Commun. ACM, vol. 51, pages 17–18, ACM, New York, NY, USA, September 2008.

Cited on Page(s): 21.

[Greensfield06] Adam Greensfield. Everyware: The dawning age of ubiquitous computing. New Riders Publishing, 1st edition, March 2006.

Cited on Page(s): 3.

[Hardt10] Moritz Hardt & Kunal Talwar. *On the geometry of differential privacy.* In Proceedings of the 42nd ACM symposium on Theory of computing, STOC '10, pages 705–714, New York, NY, USA, 2010. ACM.

Cited on Page(s): 21.

[Hay10] Michael Hay, Vibhor Rastogi, Gerome Miklau & Dan Suciu. *Boosting the accuracy of differentially private histograms through consistency.* Proc. VLDB Endow., vol. 3, pages 1021–1032, VLDB Endowment, September 2010.

Cited on Page(s): 21.

[Hazay08] Carmit Hazay & Yehuda Lindell. *Constructions of truly practical secure protocols using standard smartcards.* In CCS, CCS '08, pages 491–500, New York, NY, USA, 2008. ACM.

Cited on Page(s): 46.

[HHS02] HHS. *Standards for privacy of individually identifiable health information.* US Department of Health and Human Services (HHS), Final Rule, 2002.

Cited on Page(s): 3, 62.

[Husek08] Christian Husek. *ELGA The Electronic Health Record in Austria.* In International Conference of Society for Medical Innovation and Technology, 2008.

Cited on Page(s): 136.

[Inan10] Ali Inan, Murat Kantarcioglu, Gabriel Ghinita & Elisa Bertino. *Private record matching using differential privacy.* In Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, pages 123–134, New York, NY, USA, 2010. ACM.

Cited on Page(s): 21.

[ISO99] ISO/IEC. *Integrated Circuit(s) Cards with Contacts - Part 7: Interindustry Commands for Structured Card Query Language (SCQL)*, 1999. Standard ISO/IEC 7816-7.

Cited on Page(s): 140.

[IT06] Fierce Health IT. *Massive data loss at HCA.* Fierce Health IT, 20th of August 2006.

Cited on Page(s): 138.

[IT08] Fierce Health IT. *GA hospital health data breach due to outsourcing error.* Fierce Health IT, 28th of September 2008.

Cited on Page(s): 138.

[Jarvinen10] Kimmo Jarvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi & Thomas Schneider. *Embedded SFE: Offloading Server and Network Using Hardware Tokens.* In Radu Sion, editeur, Financial Cryptography and Data Security, volume 6052 of *Lecture Notes in Computer Science*, pages 207–221. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-14577-3_17.

Cited on Page(s): 46.

[Jiang06] Wei Jiang & Chris Clifton. *A secure distributed framework for achieving k-anonymity.* The VLDB Journal, vol. 15, pages 316–333, Springer-Verlag New York, Inc., Secaucus, NJ, USA, November 2006.

Cited on Page(s): 44, 45.

[Jin10] Xin Jin, Nan Zhang & Gautam Das. *Algorithm-safe privacy-preserving data publishing.* In Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, pages 633–644, New York, NY, USA, 2010. ACM.

Cited on Page(s): 29.

[Jurczyk08a] Pawel Jurczyk & Li Xiong. *Privacy-preserving data publishing for horizontally partitioned databases.* In Proceeding of the 17th ACM conference on Information and knowledge management, CIKM '08, pages 1321–1322, New York, NY, USA, 2008. ACM.

Cited on Page(s): 44, 45.

[Jurczyk08b] Pawel Jurczyk & Li Xiong. *Privacy-preserving data publishing for horizontally partitioned databases.* Technical Report TR-2008-013, Emory University, Math&CS Dept., 2008.

Cited on Page(s): 44, 45.

[Jurczyk09] Pawel Jurczyk & Li Xiong. *Distributed Anonymization: Achieving Privacy for Both Data Subjects and Data Providers.* In Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security XXIII, pages 191–207, Berlin, Heidelberg, 2009. Springer-Verlag.

Cited on Page(s): 44, 45.

[Katz07] Jonathan 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.

Cited on Page(s): 46.

[Kifer09] Daniel 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.

Cited on Page(s): 29.

[Kifer10] Daniel Kifer & Bing-Rong 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.

Cited on Page(s): 22, 23.

[Kifer11] Daniel Kifer & Ashwin 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.

Cited on Page(s): 21, 29, 159.

[Korolova09] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra & Alexandros Ntoulas. *Releasing search queries and clicks privately.* In Proceedings of the 18th international conference on World wide web, WWW '09, pages 171–180, New York, NY, USA, 2009. ACM.

Cited on Page(s): 21.

[Kuhn09] Eric Kuhn. *Google unveils top political searches of 2009.* CNN, 18th of December 2009.

Cited on Page(s): 3.

[LeFevre05] Kristen LeFevre, David J. DeWitt & Raghu Ramakrishnan. *Incognito: efficient full-domain K-anonymity.* In SIGMOD, pages 49–60, New York, NY, USA, 2005. ACM.

Cited on Page(s): 25, 26, 96, 99, 100.

[LeFevre06] Kristen LeFevre, David J. DeWitt & Raghu 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.

Cited on Page(s): 25, 26, 44, 94, 96, 99, 100.

[Li07a] Ninghui Li, Tiancheng Li & Suresh 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.

Cited on Page(s): 18.

[Li07b] Ninghui Li, Mahesh V. Tripunitara & Ziad Bizri. *On mutually exclusive roles and separation-of-duty.* ACM Trans. Inf. Syst. Secur., vol. 10, ACM, New York, NY, USA, May 2007.

Cited on Page(s): 150.

[Li08] Ninghui Li & Qihua Wang. *Beyond separation of duty: An algebra for specifying high-level security policies.* J. ACM, vol. 55, pages 12:1–12:46, ACM, New York, NY, USA, August 2008.

Cited on Page(s): 150.

[Li09] Tiancheng Li & Ninghui Li. *On the tradeoff between privacy and utility in data publishing.* In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '09, pages 517–526, New York, NY, USA, 2009. ACM.

Cited on Page(s): 28.

[Li10a] Chao Li, Michael Hay, Vibhor Rastogi, Gerome Miklau & Andrew McGregor. *Optimizing linear counting queries under differential privacy.* In Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '10, pages 123–134, New York, NY, USA, 2010. ACM.

Cited on Page(s): 21.

[Li10b] Ninghui Li, Tiancheng Li & Suresh Venkatasubramanian. *Closeness: A New Privacy Measure for Data Publishing.* IEEE Trans. on Knowl. and Data Eng., vol. 22, pages 943–956, IEEE Educational Activities Department, Piscataway, NJ, USA, July 2010.

Cited on Page(s): 17, 18, 62, 105.

[Liebert08] Tricia Liebert. *Ongoing concern over Pentagon network attack.* TechRepublic, 7th of March 2008.

Cited on Page(s): 138.

[Machanavajjhala06] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer & Muthuramakrishnan Venkitasubramaniam. *ℓ-Diversity: Privacy Beyond κ-Anonymity.* In Proceedings of the 22nd IEEE International Conference on Data Engineering, ICDE '06, pages 24–, Washington, DC, USA, 2006. IEEE Computer Society.

Cited on Page(s): 15, 16, 17, 62, 100, 105.

[Machanavajjhala07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke & Muthuramakrishnan Venkitasubramaniam. *l-Diversity: Privacy beyond k-Anonymity.* ACM Trans. Knowl. Discov. Data, vol. 1, no. 1, ACM, New York, NY, USA, March 2007.

Cited on Page(s): 16, 17.

[Machanavajjhala09] Ashwin Machanavajjhala, Johannes Gehrke & Michaela Götz. *Data publishing against realistic adversaries.* PVLDB, vol. 2, no. 1, pages 790–801, VLDB Endowment, August 2009.

Cited on Page(s): 22, 62, 91, 100, 105.

[Martin07] David J. Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke & Joseph Y. Halpern. *Worst-Case Background Knowledge for Privacy-Preserving Data Publishing.* In Proceedings of the 23rd IEEE International Conference on Data Engineering, pages 126–135, 2007.

Cited on Page(s): 19, 27, 105.

[Mas02] MasterCard International. *MasterCard Open Data Storage V2.0*, 2002.

Cited on Page(s): 140.

[Mearian09] Lucas Mearian. *Obama's national health records system will be costly, daunting.* Computer World, 20th of January 2009.

Cited on Page(s): 133.

[Meyerson04] Adam Meyerson & Ryan 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.

Cited on Page(s): 25, 42, 43.

[Mishra06] Nina Mishra & Mark Sandler. *Privacy via pseudorandom sketches*. In Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '06, pages 143–152, New York, NY, USA, 2006. ACM.

Cited on Page(s): 30, 31.

[Mohammed09] Noman Mohammed, Benjamin C. M. Fung, Ke Wang & Patrick 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.

Cited on Page(s): 45.

[Mohammed10] Noman Mohammed, Benjamin C. M. Fung, Patrick C. K. Hung & Cheuk-Kwong Lee. *Centralized and Distributed Anonymization for High-Dimensional Healthcare Data*. ACM Trans. Knowl. Discov. Data, vol. 4, pages 18:1–18:33, ACM, New York, NY, USA, October 2010.

Cited on Page(s): 44.

[NAHIT08] NAHIT. *Defining Key Health Information Technology Terms*. Report to the Office of the National Coordinator for Health Information Technology, 28th of April 2008.

Cited on Page(s): 134.

[Narayanan10] Arvind Narayanan & Vitaly Shmatikov. *Myths and fallacies of "Personally Identifiable Information"*. Commun. ACM, vol. 53, pages 24–26, ACM, New York, NY, USA, June 2010.

Cited on Page(s): 21.

[Nergiz07a] Mehmet Ercan Nergiz & Chris Clifton. *Thoughts on k-anonymization*. Data Knowl. Eng., vol. 63, pages 622–645, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, December 2007.

Cited on Page(s): 25.

[Nergiz07b] Mehmet Ercan Nergiz, Chris Clifton & Ahmet Erhan Nergiz. *MultiRelational k-Anonymity*. In Proceedings of the 23rd IEEE International Conference on Data Engineering, ICDE '07, pages 1417–1421, Istanbul, Turkey, 2007.

Cited on Page(s): 105.

[Neubauer09] Thomas Neubauer & Mathias Kolb. *Technologies for the Pseudonymization of Medical Data: A Legal Evaluation.* In Proceedings of the 2009 Fourth International Conference on Systems, pages 7–12, Washington, DC, USA, 2009. IEEE Computer Society.

Cited on Page(s): 5.

[NIST01] NIST. *Advanced Encryption Standard.* National Institute of Standards and Technology (NIST), FIPS Publication 197, 2001.

Cited on Page(s): 32, 34.

[NIST08a] NIST. *The Keyed-Hash Message Authentication Code (HMAC).* National Institute of Standards and Technology (NIST), FIPS Publication 198-1, 2008.

Cited on Page(s): 39.

[NIST08b] NIST. *Secure Hash Standard.* National Institute of Standards and Technology (NIST), FIPS Publication 180-3, 2008.

Cited on Page(s): 32, 38, 39.

[Park07] Hyoungmin Park & Kyuseok Shim. *Approximate algorithms for K-anonymity.* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD '07, pages 67–78, New York, NY, USA, 2007. ACM.

Cited on Page(s): 25.

[Petersen06] Jan Petersen. *GEPJ in Denmark - A Conceptual Model for Documentation of Clinical Information in the EHR in Denmark.* In Patient Data on Demand, Nordic Workshop, 2006.

Cited on Page(s): 136.

[Pucheral01] Philippe Pucheral, Luc Bouganim, Patrick Valduriez & Christophe Bobineau. *PicoDBMS: Scaling down database techniques for the smartcard.* The VLDB Journal, vol. 10, pages 120–132, Springer-Verlag New York, Inc., Secaucus, NJ, USA, September 2001.

Cited on Page(s): 140.

[Pucheral07] Phillipe Pucheral & Shaoyi Yin. *System and Method of Managing Indexation of Flash Memory.* European Patent by Gemalto and INRIA, No. 07290567.2, 2007.

Cited on Page(s): 141.

[PWC09] PWC. *Transforming healthcare through secondary use of health data.* Health Information Technologies Executive Roundtable, PriceWaterhouseCoopers Health Industries, June 2009.

Cited on Page(s): 3.

[Quantin08] Catherine Quantin, Maniane Fassa, Gouenou Coatrieux, Gilles Trouessin & François-André Allaert. *Combining Hashing and Enciphering Algorithms for Epidemiological Analysis of Gathered Data.* Methods of Information in Medicine, vol. 45, no. 5, pages 454–458, Schattauer, 2008.

Cited on Page(s): 5.

[Rastogi06] Vibhor Rastogi, Dan Suciu & Sungho Hong. *The Boundary Between Privacy and Utility in Data Anonymization.* CoRR, vol. abs/cs/0612103, 2006.

Cited on Page(s): 21, 28.

[Rastogi07] Vibhor Rastogi, Dan Suciu & Sungho 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.

Cited on Page(s): 6, 13, 17, 19, 20, 21, 28, 49, 91, 105, 128, 160.

[Rastogi09] Vibhor Rastogi, Michael Hay, Gerome Miklau & Dan Suciu. *Relationship privacy: output perturbation for queries with joins.* In Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '09, pages 107–116, New York, NY, USA, 2009. ACM.

Cited on Page(s): 21, 22.

[Rastogi10] Vibhor Rastogi & Suman Nath. *Differentially private aggregation of distributed time-series with transformation and encryption.* In Proceedings of the 2010 international conference on Management

of data, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM.

Cited on Page(s): 21.

[Ren10] Jian Ren & Jie Wu. *Survey on anonymous communications in computer networks.* Comput. Commun., vol. 33, pages 420–431, Butterworth-Heinemann, Newton, MA, USA, March 2010.

Cited on Page(s): 46, 73.

[Rogaway03] Phillip Rogaway, Mihir Bellare & John Black. *OCB: A block-cipher mode of operation for efficient authenticated encryption.* ACM Trans. Inf. Syst. Secur., vol. 6, pages 365–403, ACM, New York, NY, USA, August 2003.

Cited on Page(s): 34.

[Rose08] David Rose. *Patients avoid NHS database blunders by keeping cards close to their chest.* The Times, 28th of December 2008.

Cited on Page(s): 133, 138.

[Rosencrance03] Linda Rosencrance. *NASA Sites Hacked.* Computer World, 17th of December 2003.

Cited on Page(s): 138.

[Rosencrance06] Linda Rosencrance. *Survey: 81% of U.S. firms lost laptops with sensitive data in the past year.* Computer World, 16th of August 2006.

Cited on Page(s): 138.

[Safran07] Charles Safran, Meryl Bloomrosen, W. Edward Hammond, Steven E. Labkoff, Suzanne Markel-Fox, Paul Tang & Don Detmer. *Toward a national framework for the secondary use of health data.* Journal of American Medical Informatics Association, vol. 14, no. 1, pages 1–9, American Medical Informatics Association, Jan-Feb 2007.

Cited on Page(s): 3.

[Saltzer75] Jerome H. Saltzer & Michael D. Schroeder. *The protection of information in computer systems.* Proceedings of the IEEE, vol. 63, pages 1278–1308, September 1975.

Cited on Page(s): 149.

[Samarati98] Pierangela Samarati & Latanya Sweeney. *Generalizing data to provide anonymity when disclosing information (abstract).* In Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS '98, pages 188–, New York, NY, USA, 1998. ACM.

Cited on Page(s): 3, 15, 16.

[Samarati01] P. Samarati. *Protecting Respondents' Identities in Microdata Release.* IEEE Trans. on Knowl. and Data Eng., vol. 13, pages 1010–1027, IEEE Educational Activities Department, Piscataway, NJ, USA, November 2001.

Cited on Page(s): 25.

[Savage11] Neil Savage. *Mining Data for Better Medicine.* Technology Review, MIT, 19th of September 2011.

Cited on Page(s): 3.

[Sevastopulo07] Demetri Sevastopulo. *Chinese hacked into Pentagon.* The Finantial Times, 3rd of September 2007.

Cited on Page(s): 138.

[Stahlberg07] Patrick Stahlberg, Gerome Miklau & Brian Neil Levine. *Threats to privacy in the forensic analysis of database systems.* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD '07, pages 91–102, New York, NY, USA, 2007. ACM.

Cited on Page(s): 145.

[Stinson05] Douglas R. Stinson. Cryptography (discrete mathematics and its applications). Chapman & Hall/CRC, 2005.

Cited on Page(s): 32, 34, 38, 39.

[Sweeney00] Latanya Sweeney. *Uniqueness of Simple Demographics in the U.S. Population (White Paper).* Carnegie Mellon University, Laboratory for International Data Privacy, 2000.

Cited on Page(s): 4.

[Sweeney02] Latanya Sweeney. *k-anonymity: a model for protecting privacy.* Int. J. Uncertain. Fuzziness Knowl.-Based Syst., vol. 10, no. 5,

pages 557–570, World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2002.

Cited on Page(s): 3, 15, 16, 25, 49, 100, 105.

[Terrovitis11] Manolis Terrovitis. *Privacy preservation in the dissemination of location data.* SIGKDD Explor. Newsl., vol. 13, pages 6–18, ACM, New York, NY, USA, August 2011.

Cited on Page(s): 159.

[Tian11] Hongwei Tian & Weining Zhang. *Extending l-diversity to generalize sensitive data.* Data Knowl. Eng., vol. 70, pages 101–126, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, January 2011.

Cited on Page(s): 24.

[Trombetta11] Alberto Trombetta, Jiang Wei, Elisa Bertino & Lorenzo Bossi. *Privacy-Preserving Updates to Anonymous and Confidential Databases.* IEEE Transactions on Dependable and Secure Computing, vol. 8, no. 4, pages 578 –587, July 2011.

Cited on Page(s): 42.

[Verykios04] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin & Yannis Theodoridis. *State-of-the-art in privacy preserving data mining.* SIGMOD Rec., vol. 33, pages 50–57, ACM, New York, NY, USA, March 2004.

Cited on Page(s): 15.

[Wang04] Ke Wang, Philip S. Yu & Sourav Chakraborty. *Bottom-Up Generalization: A Data Mining Solution to Privacy Protection.* In Proceedings of the Fourth IEEE International Conference on Data Mining, ICDM '04, pages 249–256, Washington, DC, USA, 2004. IEEE Computer Society.

Cited on Page(s): 25.

[Warner65] Stanley L. Warner. *Randomized Response: A survey technique for eliminating evasive answer bias.* Journal of the American Statistical Association, vol. 60, pages 63–69, 1965.

Cited on Page(s): 13, 30, 31.

[Weiss07] Eric Weiss. *Consultant Breached FBI's Computers.* The Washington Post, 6th of July 2007.

Cited on Page(s): 138.

[WFTV08] WFTV. *Medical Center Patient Records Posted On Internet.* WFTV, 14th of August 2008.

Cited on Page(s): 138.

[Wong07] Raymond Chi-Wing Wong, Ada Wai-Chee Fu, Ke Wang & Jian Pei. *Minimality attack in privacy preserving data publishing.* In Proceedings of the 33rd international conference on Very large data bases, VLDB '07, pages 543–554. VLDB Endowment, 2007.

Cited on Page(s): 29.

[Xiao06] Xiaokui Xiao & Yufei 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.

Cited on Page(s): 27, 99.

[Xiao07] Xiaokui Xiao & Yufei Tao. *M-invariance: towards privacy preserving re-publication of dynamic datasets.* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD '07, pages 689–700, New York, NY, USA, 2007. ACM.

Cited on Page(s): 150, 160.

[Xiao10] Xiaokui Xiao, Yufei Tao & Nick Koudas. *Transparent anonymization: Thwarting adversaries who know the algorithm.* ACM Trans. Database Syst., vol. 35, pages 8:1–8:48, ACM, New York, NY, USA, May 2010.

Cited on Page(s): 29.

[Xu06] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi & Ada Wai-Chee Fu. *Utility-based anonymization using local recoding.* In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '06, pages 785–790, New York, NY, USA, 2006. ACM.

Cited on Page(s): 25.

[Xue11] Mingqiang Xue, Panagiotis Papadimitriou, Chedy Raïssi, Panos Kalnis & Hung Keng 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.

Cited on Page(s): 43.

[Yang05] Zhiqiang Yang, Sheng Zhong & Rebecca N. Wright. *Anonymity-preserving data collection.* In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, KDD '05, pages 334–343, New York, NY, USA, 2005. ACM.

Cited on Page(s): 46.

[Yao82] Andrew 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.

Cited on Page(s): 41.

[Yao86] Andrew C. Yao. *How to generate and exchange secrets.* In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

Cited on Page(s): 41.

[Yin09] Shaoyi Yin, Philippe Pucheral & Xiaofeng Meng. *A sequential indexing scheme for flash-based embedded systems.* In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '09, pages 588–599, New York, NY, USA, 2009. ACM.

Cited on Page(s): 141.

[Zhang05] Nan Zhang & Wei 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.

Cited on Page(s): 55.

[Zhang07a] Lei Zhang, Sushil Jajodia & Alexander Brodsky. *Information disclosure under realistic assumptions: privacy versus optimality.* In Proceedings of the 14th ACM conference on Computer and communications security, CCS '07, pages 573–583, New York, NY, USA, 2007. ACM.

Cited on Page(s): 29.

[Zhang07b] Qing Zhang, Nick Koudas, Divesh Srivastava & Ting Yu. *Aggregate query answering on anonymized tables.* In Proceedings of the 23rd IEEE International Conference on Data Engineering, ICDE '07, pages 116–125, 2007.

Cited on Page(s): 27.

[Zhong05] Sheng Zhong, Zhiqiang Yang & Rebecca 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.

Cited on Page(s): 42, 43, 44, 54.

[Zhong09] Sheng Zhong, Zhiqiang Yang & Tingting Chen. *k-Anonymous data collection.* Inf. Sci., vol. 179, pages 2948–2963, Elsevier Science Inc., New York, NY, USA, August 2009.

Cited on Page(s): 43, 44.