Privacy-Preserving Query Execution using Tamper Resistant Hardware. Design and Performance Considerations

Cuong Quoc To

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

Département d'informatique
UFR de Sciences

École Doctorale "STV"

Privacy-Preserving Query Execution using Tamper Resistant Hardware

Design and Performance Considerations

THÈSE

Présentée et soutenue publiquement le mercredi 16 septembre 2015

pour l'obtention du

Doctorat de l'université de Versailles Saint-Quentin-en-Yvelines
(spécialité informatique)

par

Quoc-Cuong TO

Composition du jury

Directeur :	Philippe PUCHERAL	Professeur, Université de Versailles Saint-Quentin-en-Yvelines et INRIA Paris-Rocquencourt.
Co-directeur :	Benjamin NGUYEN	Professeur, INSA Centre-Val de Loire.
Rapporteurs :	Sihem AMER-YAHIA	Directrice de recherche, CNRS.
	Ernesto DAMIANI	Professeur, Khalifa University.
Examinateurs :	Philippe LAMARRE	Professeur, INSA Lyon.
	Sébastien GAMBS	Maitre de Conferences, IRISA / INRIA, Université de de Rennes.

Laboratoire PRISM

Informatics mathematics Inria

Abstract

Current applications, from complex sensor systems (e.g. quantified self) to online e-markets acquire vast quantities of personal information which usually end-up on central servers. This massive amount of personal data, *the new oil*, represents an unprecedented potential for applications and business. However, centralizing and processing all one's data in a single server, where they are exposed to prying eyes, poses a major problem with regards to privacy concern.

Conversely, decentralized architectures helping individuals keep full control of their data, but they complexify global treatments and queries, impeding the development of innovative services.

In this thesis, we aim at reconciling individual's privacy on one side and global benefits for the community and business perspectives on the other side. It promotes the idea of pushing the security to secure hardware devices controlling the data at the place of their acquisition. Thanks to these tangible physical elements of trust, secure distributed querying protocols can reestablish the capacity to perform global computations, such as SQL aggregates, without revealing any sensitive information to central servers.

This thesis studies the subset of SQL queries without external joins and shows how to secure their execution in the presence of honest-but-curious attackers. It also discusses how the resulting querying protocols can be integrated in a concrete decentralized architecture. Cost models and experiments on SQL/AA, our distributed prototype running on real tamper-resistant hardware, demonstrate that this approach can scale to nationwide applications.

Résumé en français

Les applications actuelles, des systèmes de capteurs complexes (par exemple auto quantifiée) aux applications de e-commerce, acquièrent de grandes quantités d'informations personnelles qui sont habituellement stockées sur des serveurs centraux. Cette quantité massive de données personnelles, considéré comme le *nouveau pétrole*, représente un important potentiel pour les applications et les entreprises. Cependant, la centralisation et le traitement de toutes les données sur un serveur unique, où elles sont exposées aux indiscrétions de son gestionnaire, posent un problème majeur en ce qui concerne la vie privée.

Inversement, les architectures décentralisées aident les individus à conserver le plein de contrôle sur leurs données, toutefois leurs traitements en particulier le calcul de requêtes globales deviennent complexes.

Dans cette thèse, nous visons à concilier la vie privée de l'individu et l'exploitation de ces données, qui présentent des avantages manifestes pour la communauté (comme des études statistiques) ou encore des perspectives d'affaires. Nous promouvons l'idée de sécuriser l'acquisition des données par l'utilisation de matériel sécurisé. Grâce à ces éléments matériels tangibles de confiance, sécuriser des protocoles d'interrogation distribués permet d'effectuer des calculs globaux, tels que les agrégats SQL, sans révéler d'informations sensibles à des serveurs centraux.

Cette thèse étudie le sous-groupe de requêtes SQL sans jointures et montre comment sécuriser leur exécution en présence d'attaquants *honnêtes-mais-curieux*. Cette thèse explique également comment les protocoles d'interrogation qui en résultent peuvent être intégrés concrètement dans une architecture décentralisée. Nous démontrons que notre approche est viable et peut passer à l'échelle d'applications de la taille d'un pays par un modèle de coût et des expériences réelles sur notre prototype, SQL/AA.

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude à Philippe Pucheral, mon directeur de thèse, et Benjamin Nguyen, co-directeur de cette thèse. Je les remercie pour l'aide et le soutien qu'ils m'ont apporté pendant ces trois années. Je leur suis très reconnaissant pour leurs conseils, leurs critiques, leurs qualités humaines et leurs encouragements qui ont contribué à l'aboutissement de cette thèse.

J'adresse mes plus vifs remerciements aux membres du jury qui ont bien voulu consacrer à ma thèse une partie de leur temps. Je cite en particulier Sihem Amer-Yahia et Ernesto Damiani qui m'ont fait l'honneur d'accepter d'être rapporteurs de ma thèse. Je remercie également Philippe Lamarre et Sébastien Gambs pour avoir accepté de faire partie de mon jury de thèse.

Ma reconnaissance va aux membres de l'équipe SMIS, qui m'ont permis de réaliser cette thèse dans des conditions privilégiées, qui font de SMIS un environnement très agréable et motivant. Un grand merci à Alexei Troussov et Quentin Lefebvre pour sa disponibilité et sa spontanéité à partager ses connaissances techniques lors de l'implémentation et des expérimentations.

Je remercie Philippe Bonnet de ses contributions précieuses pour l'article EDBT. J'adresse aussi mes remerciements à Anne Cantaut et à Matthieu Finiasz pour les conseils prodigués sur les protocoles de gestion de clés de chiffrement. Je tiens également à remercier Christophe Cérin et Nicolas Grenesche pour leur accueil et les conditions de travail pour les expériences de MapReduce sur les clusters de l'Université Paris 13, ainsi que Christian Toinard de l'INSA Centre Val de Loire pour avoir initié cette collaboration et m'avoir accueilli pour un court séjour à Bourges. Je remercie également Daniel Le Métayer, directeur de la *Collaborative Action on the Protection of Privacy Rights in the Information Society* qui a financé une partie de mes recherches.

Tous mes remerciements à ma mère et ma femme qui m'ont supporté durant ces années de thèse, à ma famille et à mes amis qui m'ont aussi bien soutenu pour franchir la dernière étape de mes études !

Table of contents

List of figures

Chapter 1
Introduction

Current applications, from complex sensor systems (e.g. quantified self) to online e-markets acquire vast quantities of personal information which usually ends-up on central servers. Decentralized architectures, devised to help individuals keep full control of their data, hinder global treatments and queries, impeding the development of services of great interest. To address this challenge, we propose secure distributed querying protocols based on the use of a tangible physical element of trust, reestablishing the capacity to perform global computations without revealing any sensitive information to central servers. Thank to the recent advances in low-cost secure hardware, mass-storage secure devices are emerging and provide a real breakthrough in the management of sensitive data. They can embed personal data and/or metadata referencing documents stored encrypted in the Cloud and can manage them under the holder's control. This thesis promotes the idea of pushing the security to the edges of applications, through the use of secure hardware devices controlling the data at the place of their acquisition. In this chapter, we first position the value of personal data in our e-society nowadays; then we list the precise objectives of the thesis. Third, we present the main contributions of this thesis. Finally, we give an illustrative scenario and the outline of this manuscript.

1.1 Personal Data & Privacy

With the convergence of mobile communications, sensors and online social networks technologies, we are witnessing an exponential increase in the creation and consumption of personal data in today's digital society. Data is being collected on who we are, whom we have relation with, where we were and will be, and what we buy, etc. Some data is freely disclosed by users. Some other is transparently

acquired by sensor systems through analog processes (e.g., GPS tracking units, smart meters, healthcare sensors) or mechanical interactions (e.g., as simple as opening a door or putting a light on). *In fine*, all this data ends up in servers. This massive amount of personal data is so valuable that the World Economic Forum calls it "the *new oil*" [WEF12] since it represents an unprecedented potential for applications and business (e.g., car insurance billing, traffic decongestion, smart grids optimization, healthcare surveillance, participatory sensing). Mining and analyzing this personal data gives us the ability to understand the human's behavior and make profit from this knowledge. For example, to enjoy the "free" services (social network, search engine, etc.) supplied by the Internet giants (Facebook, Google, etc.), users have to provide them with unlimited free access to their data, which they monetize for billions of dollars (e.g., Facebook, is valued at approximately $50 per account). Surprisingly, while oil gives a maximum return of $0.5 per year and per dollar, US companies spend $2 billion a year on third-party data about individuals, with an estimated return around $30 for $1 invested [eMarketer].

However, centralizing and processing all one's data in a single server incurs a major problem with regards to privacy concerns. As seen with the PRISM affair[1] and the Gemalto SIM card encryption hack[2], the public opinion is starting to wonder whether these new services are not bringing us closer to the science fiction dystopias, since individuals' data is carefully scrutinized by governmental agencies and companies in charge of processing it [Montjoye12]. Privacy violations also arise from negligence and attacks and no current server-based approach, including cryptography based and server-side secure hardware [Agrawal02], seems capable of closing the gap. Conversely, decentralized architectures (e.g., personal data vault), providing better control to the user over the management of her personal data, impede global computations by construction.

This thesis aims to demonstrate that privacy protection and global computation are not antagonist and can be reconciled to the best benefit of the individuals, the community and the companies. To reach this goal, this thesis capitalizes on a novel

[1] http://fas.org/irp/eprint/eu-nsa.pdf : the surveillance program of the United States National Security Agency that collects internet communications of foreign nationals from at least nine major US internet companies.
[2] The encryption keys to millions of SIM cards, used by dozens of cellular networks in the US and around the world, were stolen by the UK and US intelligence communities.

architectural approach called *Trusted Cells* [Anciaux13]. This approach capitalizes on emerging practices and hardware advances representing a sea change in the acquisition and protection of personal data. Trusted Cells push the security to the edges of the network, through personal data servers [Allard10] running on secure smart phones, set-top boxes, plug computers[3] or secure portable secure devices[4] forming a global decentralized data platform. Indeed, thanks to the emergence of low-cost secure hardware and firmware technologies like ARM TrustZone[5], a full Trusted Execution Environment (TEE) will soon be present in any client device. In this thesis, and up to the experiments section, we consider that personal data is acquired and/or hosted by secure devices but make no additional assumption regarding the technical solution they rely on.

Global queries definitely make sense in this context. Typically, it would be helpful to compute aggregates over smart meters without disclosing individual's raw data (e.g., *compute the mean energy consumption per time period and district*). Identifying queries also make sense assuming the identified subjects consent to participate (e.g., *send an alert to people older than 80 and living in Paris if the number of people suffering from flu in* France *has reached a given threshold*). Computing SQL-like queries on this distributed infrastructure leads to two major and different problems: computing joins between data hosted at different locations and computing aggregates over this same data. This thesis addresses the second issue: how to compute global queries over decentralized personal data stores while respecting users' privacy? Indeed, we believe that the computation of aggregates is central to the many novel privacy preserving applications such as smart metering, e-administration, etc.

1.2 A decentralized, secure, and general approach

We address in this thesis the problem of answering SQL queries on a distributed infrastructure with strong guarantees of security. To this end, we suggest a radically different way of computing SQL queries with three main objectives:

[3] http://freedomboxfoundation.org/
[4] http://www.gd-sfs.com/portable-security-secure device
[5] http://www.arm.com/products/processors/technologies/trustzone.php

1. **Decentralization**: each individual manages his own data, under his control, and participates voluntarily in a survey. Hence, the assumption of the trusted central server is not necessary anymore.
2. **Security**: the protocol ensures that adversary cannot get sensitive data. The only information that an adversary can get is a set of encrypted tuples, which does not represent any benefit for him.
3. **Generality**: the protocol must scale up to nationwide dataset and must not rely on a 24/7 availability of all participants.

Our objective is to make as few restrictions on the computation model as possible. We model the information system as a global database formed by the union of a mutitude of distributed local data stores (e.g., nation-wide context) and we consider regular SQL queries (without external joins involving data from different data stores) and a traditional access control model. Hence the context we are targeting is different and more general than, (1) querying encrypted outsourced data where restrictions are put on the predicates which can be evaluated [Agrawal04, Amanatidis07, Popa11, Hacigümüs04], (2) performing privacy-preserving queries usually restricted to statistical queries matching differential privacy constraints [Fung10, Fayyoumi10] and (3) performing Secure-Multi-Party (SMC) query computations which cannot meet both query generality and scalability objectives [Kissner05].

1.3 Contributions

The contributions of this thesis are summarized as follows:

1) We propose different secure query execution techniques to evaluate regular SQL “group by” queries over a set of distributed trusted personal data stores, and study the range of applicability of these techniques.

2) We show how these techniques can be integrated in a concrete decentralized architecture.

3) We demonstrate that our approach is compatible with nation-wide contexts through a thorough analysis of cost models and performance measurements of a prototype running on real secure hardware devices.

4) We apply our protocol to MapReduce to support the security aspect of this framework.

In the first contribution of this thesis, we try to explore the design space of the protocols by applying a variety of the encryption schemes corresponding to each protocol. Then we compare these protocols to see in which scenario each suits best. To put these protocols into practice, we integrate them into a concrete architecture, leading to the second contribution. In the next contribution, cost models are proposed for each method. After conducting the unit test on a development device, we calibrate the result of this test to the cost models and compare the performance among protocols. To verify the accuracy of the cost models, the prototype running on real secure hardware devices are also implemented and its results are compared with that of cost models to compute the error rate. Finally, we show that our protocol can be applied to the MapReduce to support the security aspect of this framework.

1.4 Illustrative Context

To give the reader an overview of our system, this section gives a concrete context illustrating the challenges we tackle and their importance.

In France, there are currently 35 million electricity meters, including 20 million mechanical meters, and 15 million electronic meters. Modernization of electricity meters is a legal obligation imposed by the European Commission. In a directive of 2006, they required the meters to be "smart" by 2020. In other words they must allow users to control their consumption. The full nationwide rollout of 35 million smart meters was set to be completed by the year 2020, with an investment of €5 billion. To comply with this requirement, and in conjunction with the Energy Regulation Commission, Electricité Réseau Distribution de France (ERDF) is implementing a plan to modernize its 35 million electricity meters nationwide[6]. Those meters will generate much more detailed data on energy consumption.

To reflect the extra granularity of the data, smart meter suppliers must comply with a range of privacy requirements that go beyond what are required under the Data

[6] http://www.erdf.fr/medias/dossiers_presse/DP_ERDF_210610_1_EN.pdf

Protection Act. Those requirements, imposed as licensing conditions, mean that energy suppliers must obtain consumers' consent to collect and use consumption data at a level of granularity more detailed than daily reads or to use consumption data for marketing purposes. The suppliers can, under the framework, access consumption data up a daily level detail but consumers must be given the opportunity to opt out of that data collection.

Apparently, the challenge lies in the contradictory benefit of both parties. On one side, ERDF wants to get as much information about electrical usage of residents as it can so that it can provide its customers with better services and attractive tariff. On the other side, clients do not want to give out so much information about their electrical consumption since it can reveal their privacy (e.g., at the 1HZ granularity provided by the French Linky power meters, most electrical appliances have a distinctive energy signature. It is thus possible to infer from the power meter data inhabitants activities [Lam07]).

1.5 Outline

This thesis is composed of four main parts. The first part includes Chapter 2 and Chapter 3. Chapter 2 presents the background knowledge necessary to understand the approaches 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 the participants, and the way we propose to securely execute the SQL queries on the proposed architecture.

The second part contains Chapter 4 that details the design of our proposed protocols and analysis of their correctness and security.

The third part is composed of Chapters 5 and 6 which focus on the implementation and performance evaluation. Chapter 5 concentrates on the implementation issues such as access control, fault tolerance, load balance, and key management. This chapter also presents the prototype SQL/AA. Then, in Chapter 6, we build an analytical cost model to analyze and compare the performance among protocols. We also further evaluate the accuracy of the proposed cost model by verifying experimentally on real hardware.

The fourth part is Chapter 7 in which we apply one of our protocols to the Map Reduce to extend the security aspect of this framework.

Finally, Chapter 8 concludes and proposes some ideas for future work.

Chapter 2
Background Knowledge & Related Works

This chapter provides the necessary background knowledge to understand the contributions of this thesis. We start by introducing the StreamSQL and the Group By clauses in SQL query. Then, we give the background knowledge required for understanding the cryptographic primitives used in this work. Specifically, we focus on the properties and characteristics of deterministic and probabilistic encryptions, the two main encryption schemes used in our protocols. Next, we focus on the ways to defend against frequency-based attacks. We also explore other kinds of encryptions such as homomorphic encryption, order-preserving encryption. Finally, we overview the approaches related to this thesis. We explain why the current outsourced database services cannot meet both the performance and security requirements. We also point out the limitations of secure multi-party computation and statistical database in terms of efficiency and security in our context. We finally survey the related approaches that address different security aspects of other frameworks.

2.1 Group By SQL Query & StreamSQL

Structured Query Language (SQL) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). A query in SQL can consist of up to six clauses as follow:

```
SELECT <ATTRIBUTE AND FUNCTION LIST>
FROM <TABLE LIST>
[WHERE <CONDITION>]
[GROUP BY <GROUPING ATTRIBUTE(S)>]
[HAVING <GROUP CONDITION>]
[ORDER BY <ATTRIBUTE LIST>];
```

Answering the queries with only Select-From-Where clauses is quite simple. So, this thesis deals with a more challenging problem: computing aggregate functions (i.e., including the Group By clause) in a distributed manner. In this section, we focus on the Group By clauses.

In SQL, an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to return a single value. SQL offers several aggregate functions as follows:

- MAX: Compute the maximum element of some data set.
- MIN: Compute the minimum element of some data set.
- COUNT: Compute the number of elements in some data set.
- SUM: Compute the sum of all values in some data set.
- AVG: Compute the average of all values in some data set.
- VAR: Compute the variance of all values in some data set.
- RANK(x): Compute the rank of a given element x in some data set.
- MEDIAN: Compute the median element of some data set.
- SMALLEST(k): Given a parameter k, compute the k^{th} smallest element of some data set.
- LARGEST(k): Given a parameter k, compute the k^{th} largest element of some data set.
- DISTINCT: Compute the number of distinct elements in some data set.
- MODE: Compute the element that occurs most often in some data set.

All functions mentioned, and combinations thereof, cover a wide range of reasonable aggregation queries. Moreover, all discussed aggregate functions are traditionally categorized into three classes [Locher09]: Distributive Aggregate Function, Algebraic Aggregate Function, and Holistic Aggregate Function.

Aggregate functions belonging to the first class are called distributive. Given a partition $S_1, \ldots, S_n$ of S, a distributive aggregate function f has the property that the aggregates $f(S_1), \ldots, f(S_n)$ can be used to compute f(S). Formally, distributive aggregate functions are defined as follows.

Definition (Distributive Aggregate Function). Let S be a multiset and let $S_1, \ldots, S_n$ be a partition of S. An aggregate function f is called distributive if there is an aggregate function g such that $f(S) = g(f(S_1), \ldots, f(S_n))$.

As the name suggests, distributive aggregate functions can easily be computed distributively since partial solutions can be combined by means of a function g. Distributive aggregate functions are for example COUNT, MAX, MIN, SUM, and RANK. Apart from COUNT and RANK, it holds for these functions that the function g that joins the partial aggregates together is the same as the function f (For example, $MAX(S) = MAX(MAX(S_1),\ldots,MAX(S_n)))$. For the aggregate function COUNT the function g is simply the aggregate function SUM. If we only consider the multisets $S'_1, \ldots, S'_r$ that contain element x, the rank of x in S is $RANK(x, S) = SUM(RANK(x, S'_1),\ldots,RANK(x, S'_r))-r + 1$.

The second class of aggregate functions consists of the functions that can be computed by combining distributive aggregate functions. If f(S) can be derived from the results of distributive aggregate functions for any multiset S, then f is referred to as an algebraic aggregate function.

Definition (Algebraic Aggregate Function). An aggregate function f is called algebraic, if it can be computed with a fixed number of distributive aggregate functions.

The function AVG, which computes the average of all elements in S, is an algebraic aggregate function. Once SUM and COUNT have been computed, we get the average value by simply dividing these values. The function VAR is an algebraic aggregate function as well.

Algebraic aggregate functions are by definition not (much) harder to compute than distributive aggregate functions. In both cases it is possible to exploit the fact that sub-aggregates can be merged into the desired aggregate value. The third class distinguishes itself quite clearly from the other classes in this regard. An aggregate function is said to be holistic if it is not possible to combine sub-aggregates.

Definition (Holistic Aggregate Function). An aggregate function f is called holistic, if there is no constant bound on the size of the storage needed to describe a sub-aggregate.

Intuitively, a holistic aggregate function is a function that can only be computed by looking at each element individually. Since all functions that cannot be computed by combining sub-aggregates are considered holistic, the classification of aggregate functions into these three categories is exhaustive. The remaining aggregate functions, i.e., MEDIAN, SMALLEST(k), LARGEST(k), DISTINCT, and MODE, all belong to this class.

The fact that sub-aggregates cannot be used directly to compute the final aggregate entails that holistic functions are considerably more difficult to compute than distributive and algebraic aggregate functions.

In our architecture, each secure device computes part of the aggregate function. In order to compute the holistic functions, all data must be gathered in one place and then comparing each element individually. In other words, the holistic aggregate functions cannot be easily computed in a distributed way and therefore it does not fit on our distributed architecture. So, in this work, we focus on the distributive and algebraic aggregate functions and let holistic ones for future work.

StreamSQL [StreamSQL15] is a query language that extends SQL with the ability to manipulate real-time data streams, which are infinite sequences of tuples that are not all available at the same time. They are essentially all SQL extensions that incorporate a notion of a window on a stream as a way to convert an infinite stream into a finite relation in order to apply relational operators. In other words, a stream can be windowed to create finite sets of tuples (e.g., a window of size 5 minutes would contain all the tuples in a given 5 minutes period). Because of this extended feature, a StreamSQL query can be in this form:

```
SELECT STREAM [ALL | DISTINCT] select_expr,
Analytic_function(select_expr) [OVER] window_als
FROM stream_reference
[WHERE where_condition]
[GROUP BY col_list]
[WINDOW window_als AS (RANGE)]
```

Queries of interest. We consider that local databases hosted by distributed devices conform to a common schema which can be queried in SQL. For example, power meter data (resp., GPS traces, healthcare records, etc) can be stored in one or several table(s) whose schema is defined by the national distribution company (resp., an insurance company consortium, the Ministry of Health, etc) horizontally

partitioned on the local stores. Queries are regular SQL queries, borrowing the SIZE clause from the definition of windows in the StreamSQL standard as mentioned above. For example, an energy distribution company could issue the following query on customers' smart meters.

```
SELECT C.district, AVG(Cons)
FROM Power P, Consumer C
WHERE C.accomodation='detached house'
      and C.cid = P.cid
GROUP BY C.district
HAVING Count(distinct C.cid) > 100
SIZE current_date() <= 2014-04-1
```

This query computes the mean energy consumption of people living in a detached house, grouped by district, for districts where over 100 consumers answered the poll. The poll is open until the 1st of April 2014.

In the example presented above, only the smart meter of customers who opt-in for this service will participate in the computation. Needless to say that the querier, that is the distribution company, must be prevented from seeing the raw data of its customers for privacy concerns. In terms of **privacy protection**, the querying protocol must guarantee that (1) the Querier gains access only to the final result of authorized queries, as in a traditional database systems and (2) intermediate results stored in SSI are fully obfuscated. The first requirement follows the regular access control model in which each Querier with appropriate privileges is granted access to specific views of the database. These views are the results of the SQL queries. In our context, they are the results of the Group By StreamSQL queries, meaning that the Queriers can see only the final aggregated results of the queries (but not the raw data of each participant).

2.2 Cryptographic tools

As stated in chapter 1, the objective of this thesis is to perform global computations by hiding the sensitive information from untrusted servers. Whatever the architectures and solutions proposed, personal data need to be externalized and therefore must be protected by cryptographic tools. As seen in following chapters, corresponding to each encryption scheme (and therefore the level of security), there are different types of computations that can be done on them and leads to different

performance. Due to their important role, various types of encryption schemes are used in this thesis. We review these kinds of encryptions in this section.

2.2.1 Deterministic Encryption and Frequency-based Attacks

Deterministic Encryption

The first, also the simplest one, is the deterministic encryption scheme [Bellare07] that always produces the same ciphertext for a given plaintext and key, even over separate executions of the encryption algorithm. Examples of deterministic encryption algorithms include the RSA cryptosystem (without encryption padding), and many block ciphers when used in ECB mode.

Formal definition of deterministic encryption: A deterministic encryption scheme Π = (K, Ɛ, Ɗ) is specified by three polynomial-time algorithms (i.e., Key Generation, Encryption, Decryption) as follows.

Key Generation (sk,pk) <- $K(1^k)$: on input a security parameter k expressed in the unary representation 1^k, the key generation algorithm outputs a public key pk and a matching secret key sk. The pk includes a description of finite message space M and a finite ciphertext space C.

Encryption c <- Ɛ(pk,m): on input pk and a message m ϵ M, the deterministic encryption algorithm Ɛ outputs a ciphertext c ϵ C.

Decryption m <- Ɗ(sk,c): on input a secret key sk and a ciphertext c, the decryption algorithm outputs a message m ϵ M.

While deterministic encryption permits logarithmic time search on encrypted data, it is easy to detect if a message is sent twice, opening the door for frequency-based attacks.

Frequency-based attacks

The frequency-based attack is a type of attack that exploits additional adversary knowledge of domain values and/or their exact/approximate frequencies to crack the encrypted data. To cope with frequency-based attacks, the straightforward 1-to-1 substitution encryption functions (e.g., deterministic encryption) are not sufficient. For

example, to protect user privacy in location-based services, their locations will be encrypted in the storage. However, a 1-to-1 encryption scheme on locations is not secure, as the attacker can map the encrypted data values of the highest frequency to the popular locations easily. In reality, the attacker may possess approximate knowledge of the frequencies or may know the exact/approximate supports of a subset of data values in the network.

If the attacker knows the exact frequency of plaintext data values and utilizes such knowledge to crack the data encryption by matching the encrypted data values with original data values based on their frequency distribution. Therefore, our data encryption strategy aims to transform the original frequency distribution of the original data (i.e., plaintext) to a uniform distribution of the encrypted data (i.e., ciphertext) so that the attacker cannot derive the mapping relationship between encrypted data and original data based on her knowledge of domain values and their occurrence frequency.

Previous works [Wong07; Molloy09] consider how to defend against the frequency-based attack in the data-mining-as-service paradigm (i.e., the data mining computations are outsourced to a third-party service provider). For example, Wong et al. [Wong07] propose a substitution cipher technique on transactional data for secure outsourcing of association rule mining. It deploys a one-to-n item mapping that transforms transactions non-deterministically. However, the mapping scheme has potential security flaws; Molloy et al. [Molloy09] introduce an attack that could break the encoding scheme in [Wong07]. Some other works [Wang06; Agrawal04] consider the frequency-based attack in the scenario of the database-as-service paradigm. The basic idea is to transform the dataset in a way that, no matter what the frequency distribution of the cleartext dataset is, the ciphertext values always follow some given target distribution. Therefore, the attacker cannot decide the mapping relationship between plaintext and ciphertext values by the frequency of plaintext and ciphertext values. Wang and Lakshmanan [Wang06] propose an approach that could transform the original occurrence frequency distribution of plaintext into a uniform distribution. Agrawal et al. [Agrawal04] proposes to transform the original occurrence frequency distribution to a certain target distribution, such as Gaussian distribution. However, all of these works coped with the frequency-based attack in a centralized framework; none of the works can be applied directly to distributed data storage of wireless networks.

This kind of attack inspired the development of probabilistic encryption schemes by Goldwasser and Micali [Goldwasser84].

2.2.2 Probabilistic (Non-deterministic) Encryption

Probabilistic encryption is the use of randomness in an encryption algorithm, so that when encrypting the same message several times it will, in general, yield different ciphertexts. The first provably-secure probabilistic public-key encryption scheme was proposed by Shafi Goldwasser and Silvio Micali, based on the hardness of the quadratic residuosity problem and had a message expansion factor equal to the public key size. Example of probabilistic encryption using any trapdoor permutation:

$Enc(x) = (f(r), x \text{ XOR } b(r))$

$Dec(y, z) = b(f^{-1}(y)) \text{ XOR } z$

With x - single bit plaintext; f - trapdoor permutation (deterministic encryption algorithm); b - hard core predicate of f; r - random string

Example of probabilistic encryption in the random oracle model:

$Enc(x) = (f(r), x \text{ XOR } h(r))$

$Dec(y, z) = h(f^{-1}(y)) \text{ XOR } z$

With h being random oracle (typically implemented using a publicly specified hash function).

Deterministic encryption permits logarithmic time search on encrypted data, while probabilistic encryption only allows linear time search [Boneh04, Song00], meaning a search requires scanning the whole database. This difference is crucial for large outsourced databases which cannot afford to slow down search. Of course deterministic encryption cannot achieve the classical notions of security of randomized encryption due to its inability to hide the original frequency distribution of the plaintext domain, especially if the plaintext domain has the skewed frequency distribution. Deterministic encryption leaks equality and is only semantically secure if it can ensure that the way the data is structured prevents redundant information (e.g., if the original frequency distribution is uniform). For example, to encrypt user

information, the user's id and username would be encrypted deterministically to allow fast retrieval on these attributes; the rest of their information would be encrypted probabilistically. Since user's ids and usernames are always unique, the adversary cannot derive any knowledge (besides length / block size) from the encryptions.

Despite its high security due to its randomness, it is impossible to perform computation on non-deterministically encrypted data without decrypting it (also because of its randomness). Related works [Agrawal04, Gentry09, Hacigumus02] introduce some encryption schemes and obfuscation techniques that allow operations to be performed on encrypted data as if it were still in its plaintext form. We review these encryption schemes in the next section.

2.2.3 Other Encryption Scheme

Beside the deterministic and non-deterministic encryptions which are the two principal encryptions used in our thesis, there are some other kinds of encryptions that can help compute directly on encrypted data to some extent.

Order-preserving encryptions

Order-preserving encryptions [Agrawal04, Boldyreva09] are deterministic encryption schemes whose encryption function preserves numerical ordering of the plaintexts. The reason for interest in such schemes is that they allow efficient range queries on encrypted data.

Formally, for A,B $\subseteq$ N with |A| $\leq$ |B|, a function f : A $\rightarrow$ B is order-preserving if for all i, j $\in$ A, f(i) > f(j) iff i > j. We say that deterministic encryption scheme SE = ($\mathcal{K}$, Enc, Dec) with plaintext and ciphertext-spaces D, R is order-preserving if Enc(K, $\cdot$) is an order-preserving function from D to R for all K output by $\mathcal{K}$ (with elements of D,R interpreted as numbers, encoded as strings).

Homomorphic encryption

The homomorphic encryption [Gentry09] is a form of encryption where one can perform a specific algebraic operation on the plaintext by performing a (possibly different) algebraic operation on the ciphertext.

Informally speaking, a homomorphic cryptosystem is a cryptosystem with the additional property that there exists an efficient algorithm to compute an encryption of a function, of two messages given the public key and the encryptions of the messages but not the messages themselves.

Formal definition of Homomorphic Property: A is an algorithm that on input 1^k, k_e, and elements $c_1,c_2 \in C$ outputs an element $c_3 \in C$ so that for all $m_1,m_2 \in M$ it holds: if $m_3 = m_1 \circ m_2$ and $c_1=E(1^k,k_e,m_1)$, and $c_2=E(1^k,k_e,m_2)$, then $Prob[D(A(1^k,k_e,c_1,c_2))] \neq m_3]$ is negligible.

There are two kinds of homomorphic cryptosystems: partially and fully homomorphic encryption.

Partially Homomorphic Encryption: allow homomorphic computation of some operations on ciphertexts (e.g., additions, multiplications, quadratic functions, etc.).

Fully Homomorphic Encryption: A cryptosystem that supports arbitrary computation on ciphertexts is known as fully homomorphic encryption and is far more powerful. Such a scheme enables the construction of programs for any desirable functionality, which can be run on encrypted inputs to produce an encryption of the result. However, the performance of fully homomorphic encryption is still a big problem and therefore it cannot be applicable in real applications [Tu13].

Bucketization-based techniques

Besides encrypting the actual data using some semantically secure encryption algorithm like AES or homomorphic encryption, an alternative approach is to use data partitioning (also known as bucketization in the literature). Bucketization can be seen as a generalized partitioning algorithm that induces indistinguishability among data in a controlled manner. Here, the data are first partitioned into buckets and the bucket-id is set as the tag for each data item in the bucket. Hacigumus et al. [Hacigumus02] were the first ones to propose the bucketization-based data representation for query processing in an untrusted environment. Their bucketization was simply a data partitioning step similar to those used for histogram construction (e.g., equi-depth, equi-width partitioning, etc.) followed by assignment of a random (index) tag to each bucket effectively making every element within a bucket indistinguishable from another.

Generally speaking, a histogram on attribute is constructed by partitioning the data distribution D into mutually disjoint β subsets called buckets and approximating the frequencies f and values V in each bucket in some common fashion. The simplest type of histograms is the traditional equi-width histogram, in which the input value range is subdivided into buckets having the same width, and then the count of items in each bucket is reported. Knowing the minimum and maximum values of the data, the equi-width histograms are the easiest to implement both in databases and in data streams. However, for many practical applications, such as fitting a distribution function or optimizing queries, equi-width histograms may not provide useful enough information [Greenwald96]. A better choice for these applications is an Equi-depth histogram [Greenwald96, Muralikrishna88] (also known as equi-height or equi-probable) in which the goal is to partition data into buckets such that the number of tuples in each bucket is the same. This type of histograms is more effective than equi-width histograms particularly for the data sets with skewed distributions [Ioannidis03].

Other types of histograms proposed in the literature include the following: (i) V-Optimal Histograms [Guha01, Jagadish98] that estimate the ordered input as a step-function (or pairwise linear function) with a specific number of steps, (ii) MaxDiff histograms [Poosala96] which aim to find the B − 1 largest gaps (boundaries) in the sorted list of input, and (iii) Compressed histograms [Poosala96] which place the highest frequency values in singleton buckets and use equi-width histogram for the rest of input data. This third type can be used to construct biased histograms [Cormode06]. Although these type of histograms can be more accurate than the other histograms, they are more expensive to construct and update incrementally [Halim09].

2.3 Related Works on Querying Outsourced Databases

This work has connections with related studies in different domains, namely protection of outsourced (personal) databases, statistical databases and secures aggregation in sensor networks. We review these works below.

2.3.1 Querying Encrypted Databases

Outsourced database services or Database-as-a-Service (DaaS) [Hacigumus02] allow users to store sensitive data on a remote, untrusted server and retrieve desired parts of it on request. Many works have addressed the security of DaaS by encrypting the data at rest and pushing part of the processing to the server side. Searchable encryption has been studied in the symmetric-key [Amanatidis07] and public-key [Bellare07] settings but these works focus mainly on simple exact-match queries and introduce a high computing cost. Agrawal *et al.* [Agrawal04] proposed an order preserving encryption scheme (OPES), which ensures that the order among plaintext data is preserved in the ciphertext domain, supporting range and aggregate queries, but OPES relies on the strong assumption that all plaintexts in the database are known in advance and order-preserving is usually synonym of weaker security. The assumption on the *a priori* knowledge of all plaintext is not always practical (e.g., in our highly distributed database context, users do not know all plaintexts *a priori*), so a stateless scheme whose encryption algorithm can process single plaintexts on the fly is more practical.

Bucketization-based techniques [Hacigumus02, Hore12] use distributional properties of the dataset to partition data and design indexing techniques that allow approximate queries over encrypted data. Unlike cryptographic schemes that aim for exact predicate evaluation, bucketization admits false positives while ensuring all matching data is retrieved. A post-processing step is required at the client-side to weed out the false positives. These techniques often support limited types of queries and lack of a precise analysis of the performance/security tradeoff introduced by the indexes. To overcome this limitation, the work in [Damiani03] quantitatively measures the resulting inference exposure.

Other works introduce solutions to compute basic arithmetic over encrypted data, but homomorphic encryption [Paillier99] supports only range queries, fully homomorphic encryption [Gentry09] is unrealistic in terms of time, and privacy homomorphism [Hacigumus04] is insecure under ciphertext-only attacks [Mykletun06]. In terms of utility and security, the best approach would be to consider theoretical solution, the fully homomorphic encryption [Gentry09], which allows servers to compute arbitrary functions over encrypted data, while only clients see decrypted data. However, this construction is prohibitively expensive in practice, requiring slowdowns on the order

of 10^9× [Tu13]. In term of performance, CryptDB [Popa11] is a system that provides provable confidentiality by executing SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. However, this system is not completely secure since it still uses some weak encryption schemes (e.g., deterministic encryption, order-preserving encryption [Boldyreva09]).

Recently, the Monomi system [Tu13] has been proposed for securely executing analytical workloads over sensitive data on an untrusted database server. Although this system can execute complex queries with a median overhead of only 1.24× compared to an un-encrypted database, there can be only one trusted client decrypting data, and therefore it cannot enjoy the benefit of parallel computing. Another limitation of this system is that to perform the GROUP BY or equi-join queries, it still uses some weak encryption schemes (e.g., deterministic encryption). Hence, optimal performance/security tradeoff for outsourced databases is still regarded as the Holy Grail.

Some works [Bajaj11, Arasu14] deploy the secure hardware at server side to ensure the confidentiality of the system. By leveraging server-hosted tamper-proof hardware, [Bajaj11] designs TrustedDB, a trusted hardware based relational database with full data confidentiality and no limitations on query expressiveness. TrustedDB utilizes tamper resistant hardware such as the IBM 4764/5 cryptographic coprocessors deployed on the service provider's side to implement a complete SQL database processing engine. Although tamper resistant hardware provides a secure execution environment, it is significantly constrained in both computational ability and memory capacity which makes implementing fully featured database solutions using secure coprocessors very challenging. TrustedDB overcomes these limitations by utilizing resources of untrusted server to the maximum extent possible. This eliminates the limitations on the size of databases that can be supported. Moreover, client queries are pre-processed to identify sensitive components to be run inside the secure CPU. Non-sensitive operations are off-loaded to the untrusted server. However, TrustedDB does not deploy any parallel processing, limiting its performance. [Arasu14] also bases on the trusted hardware to securely decrypt data on the server and perform computations in plaintext. In this setting, since the data access pattern from untrusted storage has the potential to reveal sensitive information, they present oblivious query processing algorithms so that an adversary observing the query execution learns nothing about the underlying database.

Even equipped with secure hardware on server with strong encryption, these works do not solve the two intrinsic problems of centralized approaches. First, users get exposed to sudden changes in privacy policies by the managing infrastructures; their data can also be unexpectedly exposed by negligence or because it is regulated by too weak policies. Second, users are exposed to sophisticated attacks, whose cost-benefit is high on a centralized database [Anciaux13] (i.e., a successful attack compromises all the data stored in the centralized server while on the decentralized approaches, adversary steals only the portion of data stored in that site). In contrast, decentralized approaches return complete control of users on their data and drastically reduce the benefits/cost ratio of an attack.

2.3.2 Querying Statistical Databases

Statistical databases (SDB) [Fayyoumi10] are motivated by the desire to compute statistics without compromising sensitive information about individuals. This requires trusting the server to perform query restriction or data perturbation, to produce the approximate results, and to deliver them to untrusted queriers. Thus, the SDB model is orthogonal to our context since (1) it assumes a trusted third party (i.e., the SDB server) and (2) it usually produces approximate results to prevent queriers from conducting inferential attack [Fayyoumi10].

2.3.3 Querying Sensor Network

Wireless sensor networks (WSN) [Alzaid08] consist of sensor nodes with limited power, computation, storage, sensing and communication capabilities. In WSN, an aggregator node can compute the sum, average, minimum or maximum of the data from its children sensors, and send the aggregation results to a higher-level aggregator. WSN have some connection with our context regarding the computation of distributed aggregations. However, contrary to our context, WSN nodes are highly available, can communicate with each other in order to form a network topology to optimize calculations (In fact, secure devices can collaborate to form the topology through untrusted server, but because of the weak connectivity of secure devices, forming the topology is inefficient in term of time). Other work [Castelluccia05] uses additively homomorphic encryption for computing aggregation function on encrypted data in WSN but fails to consider queries with GROUP BY clauses. Liu *et al.* [Liu10]

protects data against frequency-based attacks but considers only point and range queries.

2.4 Other Secure Computation Frameworks

Not restricted to SQL, this section reviews the works allowing other forms of computations such as privacy-preserving data publishing, secure multi-party computation, and Map/Reduce framework.

2.4.1 Privacy-Preserving Data Publishing

Privacy-Preserving Data Publishing (PPDP) [Fung10] provides a non trusted user with some sanitized data produced by an anonymization process such as *k*-anonymity, *l*-diversity or differential privacy to cite the most common ones [Fung10]. Similarly, PPDP is orthogonal to our context since it again assumes a trusted third party (i.e., the publisher) and produces sanitized data of lower quality to match the information exposure dictated by a specific privacy model. The work in [Allard14] tackles the first limitation by pushing the trust to secure clients but keeps the objective of producing sanitized releases. Contrary to these works, our thesis targets the execution of general SQL queries, considers a traditional access control model and does not rely on a secure server.

2.4.2 Secure multi-party computation

Secure multi-party computation (SMC) allows N parties to share a computation in which each party learns only what can be inferred from their own inputs (which can then be kept private) and the output of the computation. This problem is represented as a combinatorial circuit which depends on the size of the input. The resulting cost of a SMC protocol depends on the number of inter-participant interactions, which in turn depends exponentially on the size of the input data, on the complexity of the initial function, and on the number of participants. Despite their unquestionable theoretical interest, generic SMC approaches are impractical where inputs are large and the function to be computed complex. Ad-hoc SMC protocols have been proposed [Kissner05] to solve specific problems/functions but they lack of generality and usually make strong assumptions on participants' availability. Hence, SMC is badly adapted to our context.

2.4.3 Security in MapReduce Framework

The related works address different security aspects of MapReduce as follows.

MAC and differential privacy

[Roy10] proposes the *Airavat* that integrates mandatory access control with differential privacy in MapReduce framework. Since Airavat adds noise to the output in the reduce function to achieve differential privacy, it requires that reducers must be trusted. Furthermore, the types of computation supported by Airavat are limited (e.g., SUM, COUNT). If they want to support more kinds of computation, the mappers must also be trusted. The other drawback of Airavat is that the security mechanisms, including the integrity verification mechanisms, are implemented inside the open infrastructure, that is, they are still services provided by the infrastructure. Hence, their trustworthiness (i.e. whether they are enforced as expected) should still be verified. Although Airavat does not trust the computation provider who writes the map and reduce functions, it does trust the cloud provider and the cloud computing infrastructure. Finally, they have to modify the original MapReduce framework to support the mandatory access control.

Integrity verification

In other directions, [Wei09] replicates some map/reduce tasks and assigns them to different mappers/reducers to validate the integrity of map/reduce tasks. Any inconsistent intermediate results from those mappers/reducers reveal attacks. However, even if those malicious mappers/reducers ensure the data integrity, they cannot preserve the data privacy since the mappers/reducers directly access sensitive data in cleartexts. Recent research [Ruan12] also focuses on integrity verification, but missing the data privacy. So, these works are orthogonal to ours in which we aim at protecting the data privacy.

Data anonymization

[Zhang14b] claims that it is challenging to process large-scale data to satisfy k-anonymity in a tolerable elapsed time. So they anonymize data sets via generalization to satisfy k-anonymity requirement in a highly scalable way using MapReduce. Data sets are partitioned and anonymized in parallel in the first phase,

producing intermediate results. Then, the intermediate results are merged and further anonymized to produce consistent k-anonymous data sets in the second phase.

Hybrid Cloud

In stating that the data can be classified into secure and public data, some works [Zhang11, Zhang14a] propose the hybrid cloud including the private cloud and the public cloud. The main idea is to split the task, keeping the computation on the private data within an organization's private cloud while moving the rest to the public commercial cloud. *Sedic*, proposed in [Zhang11], automatically partitions a job according to the security levels of the data and tries to outsource as much workload to the public commercial cloud as possible, given sensitive data always stay on the private cloud. However, this solution requires that reduction operations must be associative and the original MapReduce framework must be modified. Also, the sanitization approach taken by Sedic does not fit well with chained or iterative MR, may still reveal relative locations and length of sensitive data, which could lead to crucial information leakage in certain applications [Zhang14a]. To overcome this weakness, [Zhang14a] proposes tagged-MapReduce that augments each key-value pair in MR with a sensitivity tag. However, both solutions are not suitable for MapReduce job where all data is sensitive and/or data owner does not want to reveal any data.

Encrypting part of dataset

In arguing that encrypting all data sets in the Cloud is not effective, [Zhang13] proposes an approach to identify which intermediate data sets need to be encrypted while others are in cleartexts, in order to be cost-effective while the privacy requirements of data holders can still be satisfied. The main idea is that the data with high frequency of accessing will be encrypted while the others are unencrypted. This solution is not suitable for the case where all data have the same frequency of accessing or data owner does not want to reveal even a single tuple to the untrusted Cloud.

Other works supporting very specific operations

Other works support **very specific operations**. [Blass12a] searches encrypted keywords on the Cloud so that the cloud must not learn any information about the content it hosts and search queries performed. [Blass12b] presents *EPiC* to count the number of occurrences of a pattern specified by user in an oblivious manner on the untrusted cloud. In contrast to these works, our work addresses more general problems, supporting any kind of operations.

To the best of our knowledge, no state-of-the-art MapReducre works can satisfy the three requirements of security, utility, performance and our TrustedMR proposed in Chapter 7 is the first MapReduce-based proposal, that inherits the strong privacy guarantees from [To14], achieving a secure solution to process large-scale encrypted data using a large set of tamper-resistant hardware with low performance overhead.

2.5 Conclusion

In this chapter, we first overviewed SQL queries, focusing on the Group By clause. Then, StreamSQL was introduced, and the window concept of this kind of query was emphasized due to the close relation with our interested query throughout this thesis. After the listing of various types of encryption schemes which play important role in this thesis, we reviewed different domains that apply these encryption schemes to protect the data from untrusted server.

With these constraints in mind, we then survey the state of the art that use these encryption schemes to conceal the sensitive data from the untrusted server, and find that none of them could meet all the requirements of security and efficiency.

As a conclusion, and to the best of our knowledge, our work presented in this thesis is the first proposal achieving a fully distributed and secure solution to compute aggregate SQL queries over a large set of participants.

Chapter 3
Problem Statement

In this chapter, we illustrate the Trusted Data Server (TDS) vision through different scenarios motivating our approach, and present the hypothesis related to the security of TDSs and of the queries that we are interested in. Next, we describe the asymmetric architecture, the role of a supporting server in this architecture, the threat model, and define correctness and security under this threat model. Finally, we give the problem statement.

3.1 Scenarios and Queries of Interest

As discussed in [Anciaux13], trusted hardware is more and more versatile and has become a key enabler for all applications where trust is required at the edges of the network. Figure 1 depicts different scenarios where a Trusted Data Server (TDS) is called to play a central role, by reestablishing the capacity to perform global computations without revealing any sensitive information to central servers. TDS can be integrated in energy smart meters to gather energy consumption raw data, to locally perform aggregate queries for billing or smart grid optimization purpose and externalize only certified results, thereby reconciling individuals' privacy and energy providers' benefits. Green button[7] is another application example where individuals accept sharing their data with their neighborhood through distributed queries for their personal benefit. Similarly, TDS can be integrated in GPS trackers to protect individuals' privacy while securely computing insurance fees or carbon tax and participating in general interest distributed services such as traffic jam reduction. Moreover, TDSs can be hosted in personal devices to implement secure personal

[7] http://www.greenbuttondata.org/

folders like e.g., PCEHR (Personally Controlled Electronic Health Record) fed by the individuals themselves thanks to the Blue Button initiative[8] and/or quantified-self devices. Distributed queries are useful in this context to help epidemiologists performing global surveys or allow patients suffering from the same illness to share their data in a controlled manner.

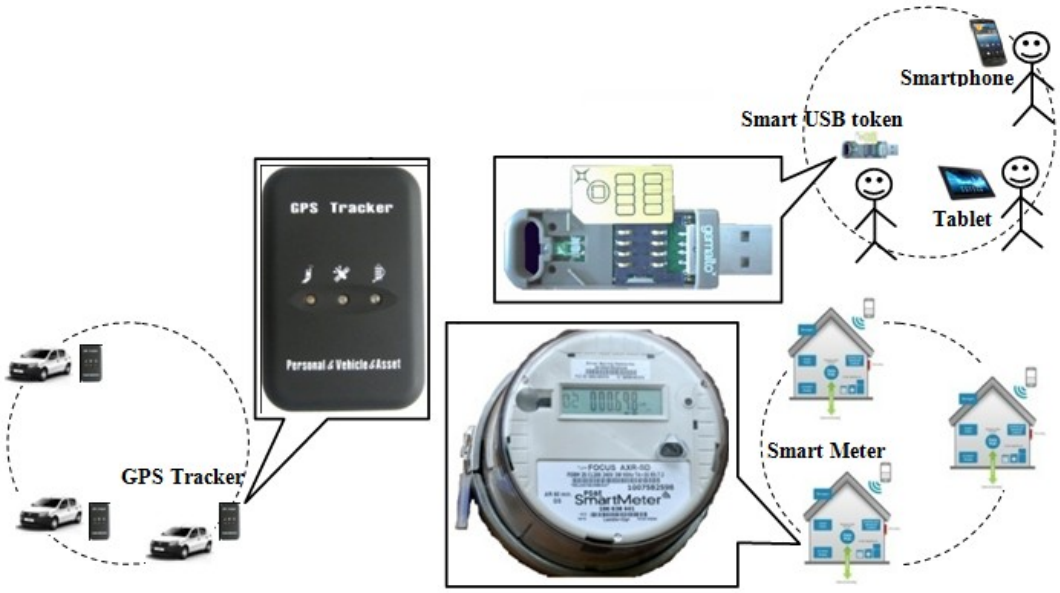


Figure 1: Different scenarios of TDSs

For the sake of generality, we make no assumption about how the data is actually gathered by TDSs, this point being application dependent [Allard10, Montjoye12]. We simply consider that *local databases* conform to a common schema (Figure 3) which can be queried in SQL. For example, power meter data (resp., GPS traces, healthcare records, etc) can be stored in one (or several) table(s) whose schema is defined by the national distribution company (resp., insurance company consortium, Ministry of Health[9], specific administration, etc). Since raw data can be highly sensitive, it must also be protected by an access control policy defined either by the producer organism, by the legislator or by a consumer association. Depending on the scenario, each individual may also opt-in/out of a particular query. For sake of generality again, we consider that each TDS participating in a distributed query protocol enforces at the same time the access control policy protecting the local data

[8] http://healthit.gov/patients-families/your-health-data
[9] This is the case in France for instance.

it hosts, with no additional consideration for the access control model itself, the choice of this model being orthogonal to this study. Hence, the objective is to let queriers (users) query this decentralized database exactly as if it were centralized, without restricting the expressive power of the language to statistical queries as in many PPDP works [Fayyoumi10**,** Popa11].

Consequently, we assume that the querier can issue the following form of SQL queries [10], borrowing the *SIZE* clause from the definition of windows in the *StreamSQL* standard [StreamSQL15]. This clause is used to indicate a maximum number of tuples to be collected, and/or a collection duration.

For example, an energy distribution company would like to issue the following query on its customers' smart meters:

```
SELECT AVG(Cons)
FROM Power P, Consumer C
WHERE C.accomodation='detached house' and C.cid = P.cid
GROUP BY C.district
HAVING Count(distinct C.cid) > 100
SIZE 50000
```

This query computes the mean energy consumption of consumers living in a detached home grouped by district, for districts where over 100 consumers answered the poll and the poll stops after having globally received at least 50.000 answers. The semantics of the query are the same as those of a stream relational query [Abadi03]. Only the smart meter of customers who opt-in for this service will participate in the computation. Needless to say that the querier, that is the distribution company, must be prevented to see the raw data of its customers for privacy concerns[11].

---

[10] As stated in the introduction, we do not consider *joins* between data stored in different TDSs in this thesis. However, joins which can be executed locally by each TDS are supported.

[11] At the 1HZ granularity provided by the French Linky power meters, most electrical appliances have a distinctive energy signature. It is thus possible to infer from the power meter data inhabitants activities [Lam *et al.* 2007].

In other scenarios where TDSs are seldom connected (e.g., querying mobile PCEHR), the time to collect the data is probably going to be quite large. Therefore the challenge is not on the overall response time, but rather to show that the query computation on the collected data is tractable in reasonable time, given local resources.

Also note that our semantics make the Open World Assumption: since we assume that data is *not* replicated over TDS, many true tuples will not be collected during the specified period and/or due to the limit, both indicated in the *SIZE* clause.

3.2 Trusted Data Server

In the context of this thesis, the records of each individual are primarily hosted by the individual's secure device. Whatever their form factor, secure devices are usually composed of a tamper-resistant micro-controller connected by a bus to a gigabytes size external secondary storage area (see Figure 2). We describe below the properties that we expect a secure device to exhibit.

High Security Guarantees.

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

- The microcontroller is tamper resistant, making hardware and side-channel attacks highly difficult.
- Software is certified according to the Common Criteria certification[12] 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 secure device'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);

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

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

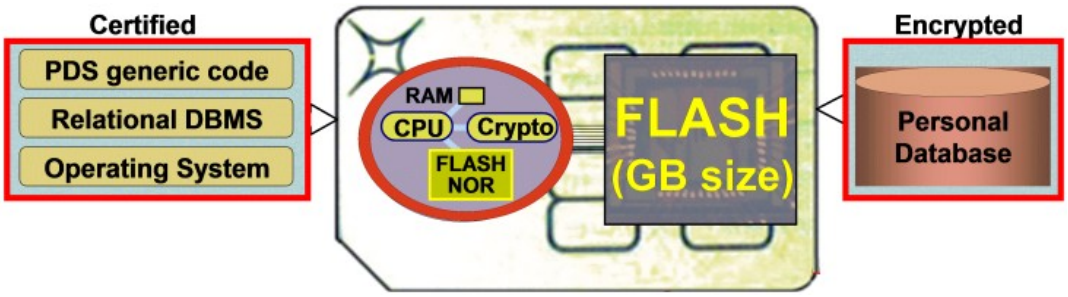


Figure 2: Trusted Data Servers

No Guarantee of Availability.

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

Modest Computing Resource.

A secure device 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 secure device and the cryptographic keys (also called cryptographic material). For the sake of simplicity, we assume that each secure device already contains its cryptographic material and privacy parameters before the protocol starts. Chapter 5 discusses practical ways of setting these pre-required data structures.

In summary, despite the diversity of existing hardware platforms, a secure device can be abstracted by (1) a Trusted Execution Environment and (2) a (potentially

untrusted but cryptographically protected) mass storage area (see Figure 2)[13]. E.g., the former can be provided by a tamper-resistant microcontroller while the latter can be provided by Flash memory. The important assumption is that the TDS code is executed by the secure device hosting it and thus cannot be tampered, even by the TDS holder herself.

3.3 Asymmetric Architecture

Now that we have described the properties that this thesis expects from the secure portable secure device, and showed the different scenarios in which the TDS plays the role. We are now ready to introduce the asymmetric architecture and its components.

3.3.1 The Role of a Supporting Server

A natural approach to tackle the problem could consist in designing a distributed protocol involving only secure devices, without any central server. They would share their data together and jointly compute the results, e.g., in a peer-to-peer fashion. A secure device 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 prohibitively high network cost (data transfers between secure devices). 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 *Supporting Server Infrastructure*, SSI for short. It is required to manage the communications between TDSs, run the distributed query protocol and store the intermediate results produced by this protocol. Because SSI is implemented on regular server(s), e.g., in the Cloud, it exhibits the same low level of trustworthiness, high computing resources, and availability. Due to these characteristics of SSI, the objective is to allow SSI to participate in the computation as much as possible to benefit its computing capability, and therefore reduce the computation load on each secure device. On the other

[13] For illustration purpose, the secure device considered in our experiments is made of a tamper-resistant microcontroller connected to a Flash memory chip.

hand, SSI is not allowed to see the sensitive information. We try to prevent SSI to obtain information as much as we can to satisfy the Security objective. All the information transferred to and stored at SSI must therefore be obfuscated appropriately.

In order to delegate the computation to the SSI, secure devices must disclose sufficient data for allowing it to compute part of the result, reducing the computation on secure devices. Besides, each secure device must handle subsets of tuples rather than the complete dataset at once to make them easily parallelizable. Thus, the resulting execution sequence consists in the following steps. In the first step, each participating secure device sends to the SSI a tuple made of its owner's record obfuscated such that the SSI can use it for grouping the appropriate data but cannot access the raw record. After grouping, in the second step, SSI sends these groups to secure devices. Finally, secure devices perform in parallel the computation on subsets of the collected dataset and return the final result. This general approach obviously ensures that the partial disclosure necessary for enabling the participation of the untrusted SSI does not thwart the privacy guarantees of the system.

In the following sections, we describe precisely the variants of this general approach, and formalize the performance and security of each variant. Three protocols are proposed and they differ in how SSI participates in the computation and which information stored at SSI, corresponding to the performance efficiency and privacy level. Intuitively, the more information exposed to SSI (and therefore the less security guarantee), the more active role of SSI in the computation, leading to the performance increase. This intuition will be formally proven in the following sections.

3.3.2 The Asymmetric Architecture

The architecture we consider is decentralized by nature. It is formed by a large set of low power TDSs embedded in secure devices. Each TDS exhibits three important properties as mention above.

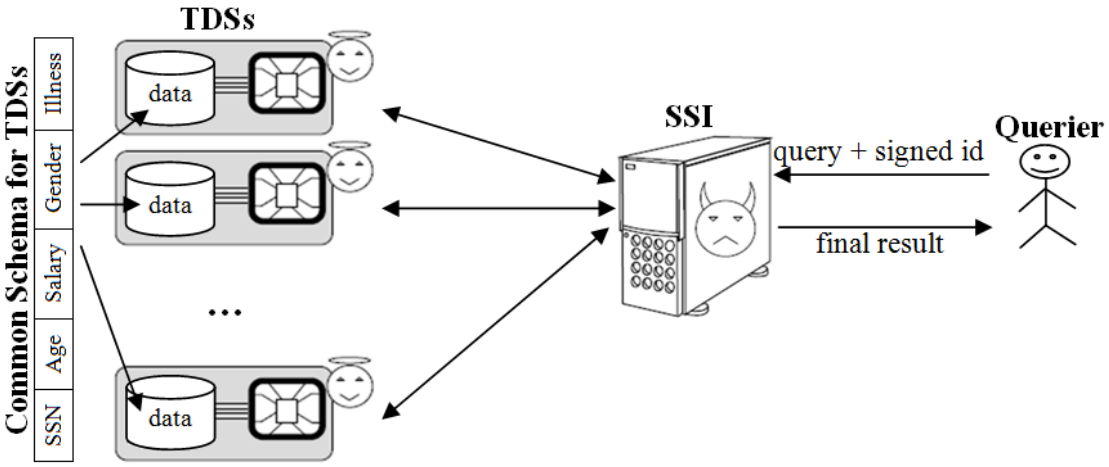


Figure 3: The Asymmetric Architecture

The computing architecture, illustrated in Figure 3, is said to be *asymmetric* in the sense that it is composed of a very large number of low power, weakly connected but highly secure TDSs and of a powerful, highly available but untrusted SSI.

3.3.3 Threat model

TDSs are the unique elements of trust in the architecture and are considered *honest*. As mentioned earlier, no trust assumption needs to be made on the TDS holder herself because a TDS is tamper-resistant and enforces the access control rules associated to its holder (just like a car driver cannot tamper the GPS tracker installed in her car by its insurance company or a customer cannot gain access to any secret data stored in her banking smartcard).

We consider honest-but-curious (also called semi-honest) SSI (i.e., which tries to infer any information it can but strictly follows the protocol). Considering malicious SSI (i.e., which may tamper the protocol with no limit, including denial-of-service) is of little interest to this study. Indeed, a malicious SSI is likely to be detected with an irreversible political/financial damage and even the risk of a class action.

The objective is thus to implement a querying protocol so that (1) the querier can gain access only to the final result of authorized queries (not to the raw data participating in the computation), as in a traditional database system and (2) intermediate results stored in SSI are obfuscated. Preventing inferential attacks by combining the result of a sequence of authorized queries as in statistical databases and PPDP work (see Chapter 2) is orthogonal to this study.

3.3.4 Correct and secure computation

Correctness:

We compute the SQL queries on the data collected during the collection phase only (i.e., when the SIZE clause is satisfied). That means that the aggregation is computed over only the subset of the population. We cannot, of course, collect all the data of the population due to the time restriction and feasibility. Then the crucial question is how we can ensure that this subset correctly reflects the whole dataset. The answer depends on many elements. The first one is the accuracy of the answer that Queriers want. Apparently, the more accurate the answer, the bigger dataset we need to collect. The second element is the distribution property of the dataset. The third element is the confident level of the final result. These elements are formulized in the Cochran's sample size formula [Cochran77]. We will delve into it in chapter 4.

Security:

In our context, the adversary is the SSI itself, and consequently accesses the intermediate data of the execution sequence stored at SSI, in addition to the encrypted output of the protocol.

In order to enable the participation of the SSI in the protocol, secure devices must disclose some controlled amount of information to it. The information voluntarily disclosed depends on each protocol to preserve the privacy guarantees (further details will be given in the following chapters).

Non-deterministic encryption is often supported for the sole purpose of protecting the data in storage and sacrifice the efficiency in execution on the encrypted data. Deterministic encryption, on the other hand, efficiently supports the execution but opens the door for frequency-based attack. So, balancing the trade-off between efficiency requirements in query execution and protection requirements due to possible inference attacks is inevitable. We investigate quantitative measures (i.e., Coefficient Exposure and Variance) to model inference exposure and provide some comparisons. These measures show how much information exposure in exchange of the efficient execution.

3.4 Problem Statement

The goal of this thesis is to design protocols to compute SQL query such that: (1) it is executed on the asymmetric architecture where security is pushed to the edge of applications, (2) its execution sequence is correct and secure, (3) where the SSI is honest-but-curious, and (4) it is scalable to datasets containing million of records.

The objective was not to find the most efficient solution for a specific problem but rather to perform a first exploration of the design space. We proposed three very different protocols, compared them according to different axes, and investigated the performance/security trade-off among protocols.

Chapter 4
The Querying Protocols

In this chapter, we first introduce the core infrastructure of our protocols, and then we show how to deal with simple SQL queries. After that, we propose protocols that can handle more complicated SQL queries, including Group By clauses. Finally, we show that our protocols are correct and secure by analyzing correctness using Cochran's model and security using the concept of coefficient exposure and variance.

4.1 Introduction

Our querying protocols share common basic mechanisms to make TDSs aware of the queries to be computed and to organize the dataflow between TDSs and queriers such that SSI cannot infer anything from the queries and their results.

Query and result delivery*:* queries are executed in pull mode. A querier posts its query to SSI and TDSs download it at connection time. To this end, SSI can maintain personal *query boxes* (in reference to mailboxes) where each TDS receives queries directed to it (e.g., *get the monthly energy consumption of consumer C*) and a global *query box* for queries directed to the crowd (e.g., *get the mean of energy consumption per month for people living in district D*). Result tuples are gathered by SSI in a temporary storage area. A query remains active until the SIZE clause is evaluated to *true* by SSI, which then informs the querier that the result is ready.

Dataflow obfuscation: all data (queries and tuples) exchanged between the querier and the TDSs, and between TDSs themselves, can be spied by SSI and must therefore be encrypted. However, an *honest-but-curious* SSI can try to conduct *frequency-based attacks* [Liu10], i.e., exploiting prior knowledge about the data distribution to infer the plaintext values of ciphertexts. Depending on the protocols

(see later), two kinds of encryption schemes will be used to prevent frequency-based attacks. With non-deterministic (aka probabilistic) encryption, denoted by *nDet_Enc*, several encryptions of the same message yield different ciphertexts while deterministic encryption (*Det_Enc* for short) always produces the same ciphertext for a given plaintext and key [Bellare07]. Whatever the encryption scheme, symmetric keys must be shared among TDSs: we note k_1 the symmetric key used by the querier and the TDSs to communicate together and k_2 the key shared by TDSs to exchange temporary results among them. Note that these keys may change over time and the way they are delivered to TDSs is discussed more deeply in chapter 5.

4.2 Select-From-Where statement

This section presents the protocol to compute Select-From-Where queries. This protocol is simple yet very useful in practice, since many queries are of this form. We also use it to help the reader get used to our approach. We tackle the more difficult Group By clause in the next section.

Let us first consider simple SQL queries of the form:

```
SELECT <attribute(s)>
FROM <Table(s)>
[WHERE <condition(s)>]
[SIZE <size condition(s)>]
```

These queries do not have a *GROUP BY* or *HAVING* clause nor involve aggregate functions in the *SELECT* clause. Hence, the selected attributes may (or may not) contain identifying information about the individuals. Though basic, these queries answer a number of practical use-cases, e.g., a doctor querying the embedded healthcare folders of her patients, or an energy provider willing to offer special prices to people matching a specific consumption profile. To compute such queries, the protocol is divided in two phases (see Figure 4):

Collection phase: (step 1) the querier posts on SSI a query Q encrypted with k_1, its credential C signed by an authority and S the SIZE clause of the query in cleartext so that SSI can evaluate it; (step 2) targeted TDSs download Q when they connect; (step 3) each of these TDSs decrypts Q, checks C, evaluates the AC policy associated to the querier and computes the result of the WHERE clause on the local

data; then each TDS either sends its result tuples (step 4), or a dummy tuple[14] whether the result is empty or the querier has not enough privilege to access these local data (step 4'), non-deterministically encrypted with k_2. The collection phase stops when the SIZE condition has been reached. The result of the collection phase is actually the result of the query, possibly complemented with dummy tuples. We call it Covering Result.

Filtering phase: (step 5) SSI partitions the Covering Result with the objective to let several TDSs manage next these partitions in parallel. The Covering Result being fully encrypted, SSI sees partitions as uninterpreted chunks of bytes; (step 6) connected TDSs download these partitions. These TDSs may be different from the ones involved in the collection phase; (step 7) each of these TDS decrypts the partition and filters out dummy tuples; (step 8) each TDS sends back the true tuples encrypted with key k_1 to SSI, which finally concatenates all results and informs the querier that she can download the result (step 9).

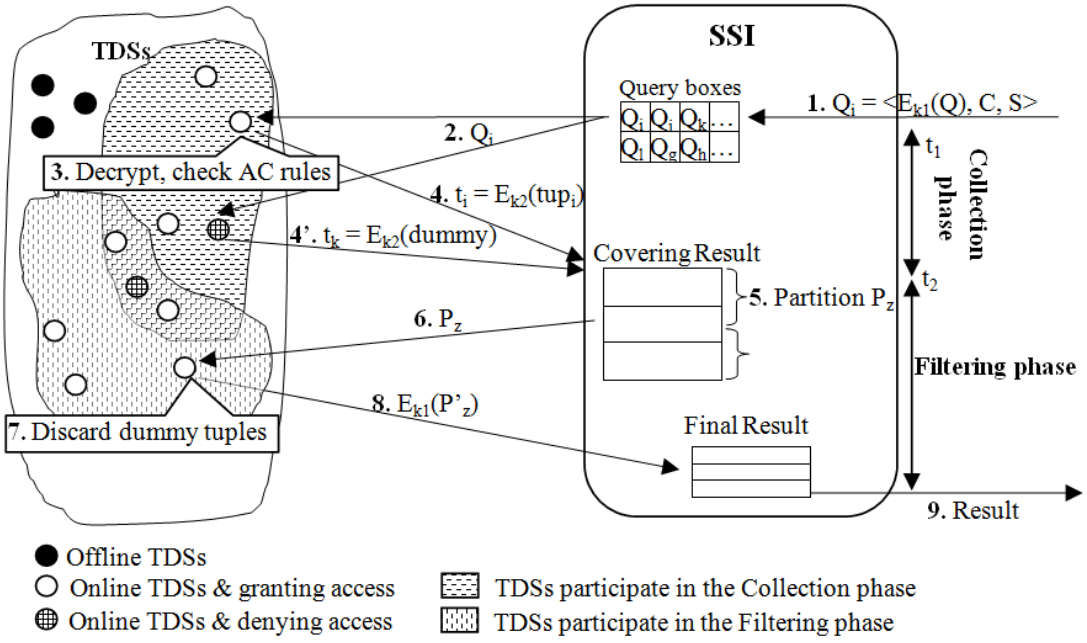


Figure 4: Select-From-Where querying protocol

[14] Even if the query is encrypted, sending dummy tuples avoids SSI to learn the query selectivity (and from that guess the query). It is also helpful in the case where SSI and querier are the same entity.

Informally speaking, the correctness, security and efficiency properties of the protocol are as follows:

Correctness. Since SSI is honest-but-curious, it will deliver to the querier all tuples returned by the TDSs. Dummy tuples are marked so that they can be recognized and removed after decryption by each TDS. Therefore the final result contains only true tuples. If a TDS goes offline in the middle of processing a partition, SSI resends that partition to another available TDS after a given timeout so that the result is complete.

Security. Since SSI does not know key k_1, it can decrypt neither the query nor the result tuples. TDSs use *nDet_Enc* for encrypting the result tuples so that SSI can neither launch any frequency-based attacks nor detect dummy tuples. There can be two additional risks. The first risk is that SSI acquires a TDS with the objective to get the cryptographic material. As stated in Chapter 3, TDS code cannot be tampered, even by its holder. Whatever the information decrypted internally, the only output that a TDS can deliver is a set of encrypted tuples, which does not represent any benefit for SSI. The second risk is if SSI colludes with the querier. For the same reason, SSI will only get the same information as the querier (i.e., the final result in clear text and no more).

Efficiency. The efficiency of the protocol is linked to the frequency of TDSs connection and to the SIZE clause. Both the collection and filtering phases are run in parallel by all connected TDSs and no time-consuming task is performed by any of them. As the experiment section will clarify, each TDS manages incoming partitions in streaming because the internal time to decrypt the data and perform the filtering is significantly less than the time needed to download the data.

While important in practice, executing Select-From-Where queries in the Trusted Cells context shows no intractable difficulties and the main objective of this section was to present the query framework in this simple context. Executing Group By queries is far more challenging. The next section will present different alternatives to tackle this problem. Rather than trying to get an optimal solution, which is context dependent, the objective is to explore the design space and show that different querying protocols may be devised to tackle a broad range of situations.

4.3 Group By Queries

The Group By clause introduces an extra phase: the computation of aggregates of data produced by different TDSs, which is the weak point for frequency-based attacks. In this section, we propose several protocols, discussing their strong and weak points from both efficiency and security points of view.

4.3.1 Generic Query Evaluation Protocol

Let us now consider general SQL queries of the form[15]:

```
SELECT <attribute(s) and/or aggregate function(s)>
FROM <Table(s)>
[WHERE <condition(s)>]
[GROUP BY <grouping attribute(s)>]
[HAVING <grouping condition(s)>]
[SIZE <size condition(s)>]
```

These queries are more challenging to compute because they require performing set-oriented computations over intermediate results sent by TDSs to SSI. The point is that TDSs usually have limited RAM, limited computing resources and limited connectivity. It is therefore unrealistic to devise a protocol where a single TDS downloads the intermediate results of all participants, decrypts them and computes the aggregation alone. On the other hand, SSI cannot help much in the processing since (1) it is not allowed to decrypt any intermediate results and (2) it cannot gather encrypted data into groups based on the encrypted value of the grouping attributes, denoted by A_G={G_i}, without gaining some knowledge about the data distribution. This would indeed violate our security assumption since the knowledge of A_G distribution opens the door to frequency-based attacks by SSI: e.g. in the extreme case where A_G contains both quasi-identifiers and sensitive values, attribute linkage would become obvious. Finally, the querier cannot help in the processing either since she is only granted access to the final result, and not to the raw data.

To solve this problem, we suggest a generic aggregation protocol divided into three phases (see Figure 5):

[15] For the sake of clarity, we concentrate on the management of distributive, algebraic and holistic aggregate functions identified in [Locher 2009] as the most prominent and useful ones.

Collection phase: similar to the basic protocol.

Aggregation phase:(step 5) SSI partitions the result of the collection phase; (step 6) connected TDSs (may be different from the ones involved in the collection phase) download these partitions; (step 7) each of these TDS decrypts the partition, eliminates the dummy tuples and computes partial aggregations (i.e., aggregates data belonging to the same group inside each partition); (step 8) each TDS sends its partial aggregations encrypted with k_2 back to SSI; depending on the protocol (see next sections), the aggregation phase is iterative, and continues until all tuples belonging to the same group have been aggregated (steps 6', 7', 8'); The last iteration produces a Covering Result containing a single (encrypted) aggregated tuple for each group.

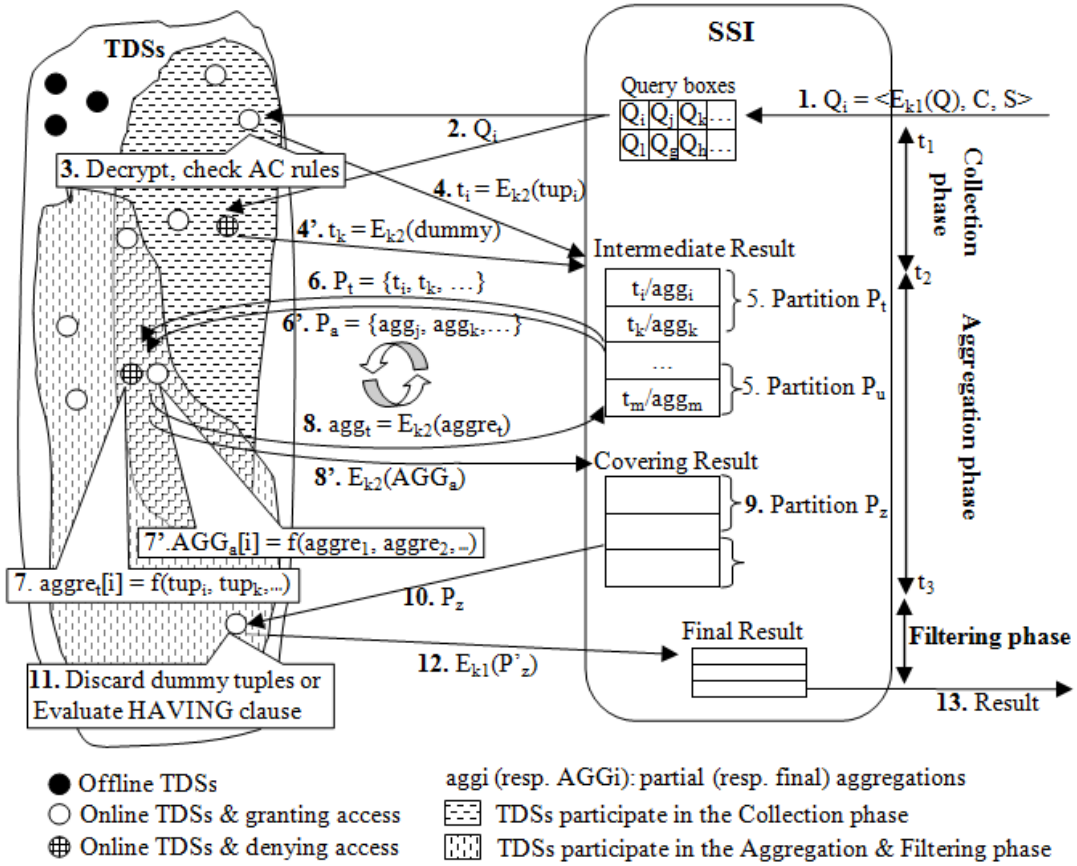


Figure 5: Group By querying protocol

Filtering phase: this phase is similar to the basic protocol except that the role of step 11 is to eliminate the groups which do not satisfy the HAVING clause instead of eliminating dummy tuples.

The rest of this section presents different variations of this generic protocol, depending on which encryption scheme is used in the collection and aggregation phases, how SSI constructs the partitions, and what information is revealed to SSI. Each solution has its own strengths and weaknesses and therefore is suitable for a specific situation. Three kinds of solutions are proposed: secure aggregation, noise-based, and histogram-based. They are subsequently compared in terms of privacy protection (Section 4.5) and performance (Chapter 6).

4.3.2 Secure Aggregation protocol

This protocol, denoted by *S_Agg* and detailed in Algorithm 1, instantiates the generic protocol as follows. In the **collection phase**, each participating TDS encrypts its result tuples using *nDet_Enc* (i.e., $nE_{k2}(tup)$) to prevent any frequency-based attack by SSI. The consequence is that SSI cannot get any knowledge about the group each tuple belongs to. Thus, during step 5, tuples from the same group are randomly distributed among the partitions. This imposes the **aggregation phase** to be iterative, as illustrated in Figure 6. At each iteration, TDSs download encrypted partitions (i.e., Ω_e) containing a sequence of (A_G, Aggregate) value pairs ((City, Energy_consumption) in the example), decrypt them to plaintext partitions (i.e., $\Omega \leftarrow nE_{k2}^{-1}(\Omega_e)$), aggregate values belonging to the same grouping attributes (i.e., $\Omega_{new} = \Omega_{old} \oplus \Omega$), and sends back to SSI a smaller encrypted sequence of (A_G, Aggregate) value pairs where values of the same group have been aggregated. SSI gathers these partial aggregations to form new partitions, and so on and so forth until a single partition (i.e., Ω_{final}) is produced, which contains the final aggregation.

Correctness. The requirement for *S_Agg* to terminate is that TDSs have enough resources to perform partial aggregations. Each TDS needs to maintain in memory a data structure called *partial aggregate* which stores the current value of the aggregate function being computed for each group. Each tuple read from the input partition contributes to the current value of the aggregate function for the group this tuple belongs to. Hence the *partial aggregate* structure must fit in RAM (or be swapped in stable storage at much higher cost). If the number of groups is high (e.g., grouping on a key attribute) and TDSs have a tiny RAM, this may become a limiting factor.

ALGORITHM 1. Secure Aggregation: S_Agg (K1, K2, Q, α)

Input: (TDS's side): The cryptographic keys *(K1, K2)*, query Q from Querier.
(SSI's side): reduction factor *α (α ≥ 2)*.
Output: The final aggregation Ω_{final}.
begin *Collection phase*
Each connected TDS sends a tuple of the form $tup_e = nE_{K2}(tup)$ *to SSI*
end
begin *Aggregation phase*
repeat
repeat
TDSs connect to SSI and SSI chooses tup_e *(or encrypted partial aggregation* Ω_e*) randomly to parallel feed these TDSs (in data stream)*
foreach *TDS ∈ TDSs* **do**
Receive tup_e *(or* Ω_e*) from SSI*
Decrypt tup_e *(or* Ω_e*):* $tup \leftarrow nE_{k2}^{-1}(tup_e)$; $\Omega \leftarrow nE_{k2}^{-1}(\Omega_e)$
Add to its partial aggregation: $\Omega = \Omega \oplus tup$; $\Omega_{new} = \Omega_{old} \oplus \Omega$
until *all* tup_e *(or* Ω_e*) in SSI have been sent to TDSs*
foreach *TDS ∈ TDSs* **do**
Encrypt its partial aggregation: $\Omega_e \leftarrow nE_{k2}(\Omega)$
Send Ω_e *to SSI*
until $n_{\Omega e} = 1$
end
Filtering phase //evaluate HAVING clause
return $nE_{k1}(\Omega_{final})$ *by SSI to Querier;*

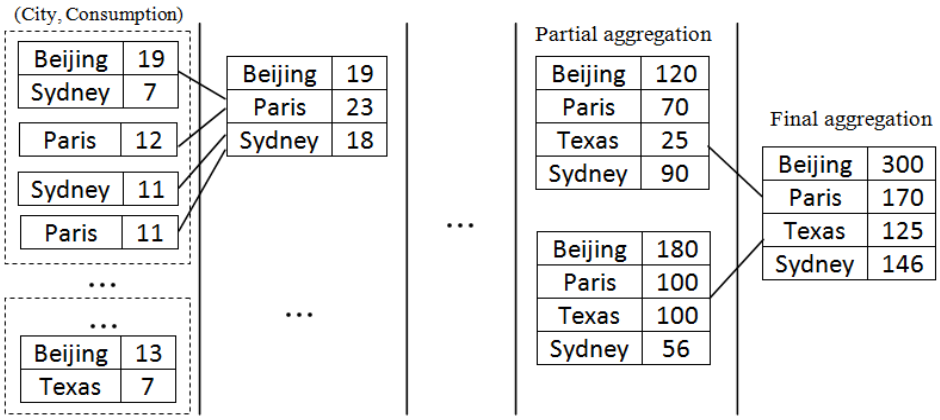


Figure 6: An example of (iterative partial) aggregation

Security. In all phases, the information revealed to SSI is a sequence of tuples or value pairs (i.e., tup_e and Ω_e) encrypted non-deterministically (*nDet_Enc*) so that SSI cannot conduct any frequency-based attack.

Efficiency. The aggregation process is such that the parallelism between TDSs decreases at each iteration (i.e., $\alpha = \frac{N_1^{TDS}}{N_2^{TDS}} = \cdots = \frac{N_{n-1}^{TDS}}{N_n^{TDS}}$, with N_i^{TDS} being the number of

TDSs that participate in the i^{th} partial aggregation phase), up to having a single TDS producing the final aggregation (i.e., N_n^{TDS} = 1). The cost model is proposed in Chapter 6 to find the optimal value for the reduction factor α. Note again that incoming partitions are managed in streaming because the cost to download the data significantly dominates the rest.

Suitable queries. Because of the limited RAM size, this algorithm is applicable for the queries with small *G* such as Q1: `SELECT AVG(Salary) FROM Paris_Population WHERE Age>20 GROUP BY Zipcode` (Paris has 20 different zip codes corresponding to 20 districts) or Q2: *`SELECT COUNT(*) FROM Paris_Population WHERE Age>20 GROUP BY Gender.`*

4.3.3 Noise-based protocols

In these protocols, called *Noise_based* and detailed in Algorithm 2, *Det_Enc* is used during the **collection phase** on the grouping attributes A_G. This is a significant change, since it allows SSI to help in data processing by assembling tuples belonging to the same groups in the same partitions. However, the downside is that using *Det_Enc* reveals the distribution of A_G to SSI. To prevent this disclosure, the fundamental idea is that TDSs add some noise (i.e., fake tuples) to the data in order to hide the real distribution. The added fake tuples must have identified characteristics, as dummy tuples, such that TDSs can filter them out in a later step. The **aggregation** phase is roughly similar to *S_Agg*, except that the content of partitions is no longer random, thereby accelerating convergence and allowing parallelism up to the final iteration. Two solutions are introduced to generate noise: random (white) noise, and noise controlled by complementary domains.

Random (white) noise solutions. In this solution, denoted R_{nf}*_Noise*, n_f fake tuples are generated randomly then added. TDSs apply *Det-Enc* on A_G, and *nDet_Enc* on $\bar{A}_G$ (the attributes not appearing in the *GROUP BY* clause). However, because the fake tuples are randomly generated, the distribution of mixed values may not be different enough from that of true values especially if the disparity in frequency among A_G is big. To overcome this difficulty, a large quantity of fake tuples (n_f>>1) must be injected to make the fake distribution dominate the true one.

ALGORITHM 2. Random noise: R_{nf}_Noise (K1, K2, Q, n_f)

Input: (TDS's side): the cryptographic keys *(K1, K2)*, query Q from Querier.
Output: The final aggregation Ω_{final}.
begin *Collection phase*
  Each connected TDS sends (n_f+1) tuples of the form tup_e = ($E_{K2}(A_G)$,$nE_{K2}(\bar{A}_G)$ to SSI
end
begin *Aggregation phase*
  repeat *//(on SSI side)*
    SSI groups tup_e with the same $E_{K2}(A_G)$
    TDSs connect to SSI to download these groups (in data stream)
  until *all groups in SSI have been sent to TDSs*
  foreach *TDS ∈ TDSs* **do** *//(on TDS side)*
    repeat
      Receive tup_e from SSI
      Decrypt tup_e: $A_G \leftarrow E_{k2}^{-1}(A_{Ge})$; $\bar{A}_G \leftarrow nE_{k2}^{-1}(\bar{A}_{Ge})$
      Filter false tuples (based on the identified characteristics)
      Compute the aggregate values for the group A_G: [A_G, AGG]
    until *no more tuples received from SSI*
    Encrypt this aggregate value: [$E_{K2}(A_G)$,$nE_{K2}(AGG)$]
    Send this encrypted aggregation to SSI
end
Filtering phase //evaluate HAVING clause
return *$nE_{k1}(\Omega_{final})$ to Querier by SSI;*

Noise controlled by complementary domains. This solution, called *C_Noise*, overcomes the limitation of *R_{nf}_Noise* by generating fake tuples based on the prior knowledge of the A_G domain cardinality. Let us assume that A_G domain cardinality is n_d(e.g., for attribute Age, $n_d \approx 130$), a TDS will generate n_d - 1 fake tuples, one for each value different from the true one. The resulting distribution is totally flat by construction. However, if the domain cardinality is not readily available, a cardinality discovering algorithm must be launched beforehand (see next section).

Correctness. True tuples are grouped in partitions according to the value of their A_G attributes so that the aggregate function can be computed correctly. Fake tuples are eliminated during the aggregation phase by TDSs thanks to their identified characteristics and do not contribute to the computation.

Security. Although TDSs apply *Det-Enc* on A_G, A_G distribution remains hidden to SSI by injecting enough white noise such that the fake distribution dominates the true one or by adding controlled noise producing a flat distribution.

Efficiency. TDSs do not need to materialize a large partial aggregate structure as in *S_Agg* because each partition contains tuples belonging to a small set of (ideally one)

groups. Additionally, this property guarantees the convergence of the aggregation process and increases the parallelism in all phases of the protocol. However, the price to pay is the production and the elimination afterwards of a potentially very high number of fake tuples (the value is algorithm and data dependent).

Suitable queries. R_{nf}*_Noise* with small n_f is suitable for the queries in which there is no wide disparity in frequency between A_G such as Q3: `SELECT COUNT(Child) FROM Paris_Population GROUP BY Father's_Name HAVING COUNT(Child) < 4.` In contrast, the white noise solution with big n_f is suitable for queries with big disparity such as Q4: `SELECT COUNT(*) FROM Paris_Population GROUP BY Salary` because the number of very rich people (i.e, salary > 1 M€/year) is much less than that of people having average salary. For the *C_Noise*, in term of the feasibility, because the process of calculating aggregation is divided among connected TDSs in a distributed and parallel way, better balancing the loads between TDSs, this protocol is applicable not only for the queries where G is small (e.g., Q1, Q2) but also for those with big G, such as Q5: `SELECT AVG(Salary) FROM Paris_Population WHERE Age>20 GROUP BY Age` (because the Age's domain is 130 at maximum). However, considering the efficiency, because the number of fake tuples is proportional to *G*, this solution is inappropriate for the queries with very big G (e.g., Q4) when it has to generate and process a large amount number of fake tuples.

4.3.4 Equi-depth histogram-based protocol

Getting a prior knowledge of the domain extension of A_G allows significant optimizations as illustrated by *C_Noise*. Let us go one step further and exploit the prior knowledge of the real distribution of A_G attributes. The idea is no longer to generate noisy data but rather to produce a uniform distribution of true data sent to SSI by grouping them into equi-depth histograms, in a way similar to [Hacigumus02]. The protocol, named *ED_Hist*, works as follows. Before entering the protocol, the distribution of A_G attributes must be discovered and distributed to all TDSs. This process needs to be done only once and refreshed from time to time instead of being run for each query. The discovery process is similar to computing a *Count* function on *Group By* A_G and can therefore be performed using one of the protocol introduced above. During the **collection phase**, each TDS uses this knowledge to calculate *nearly equi-depth histograms* that is a decomposition of the A_G domain into buckets

holding *nearly* the same number of true tuples. Each bucket is identified by a hash value giving no information about the position of the bucket elements in the domain. Then the TDS allocates its tuple(s) to the corresponding bucket(s) and sends to SSI couples of the form (*h*(bucketId), *nDet_Enc*(tuple)). During the partitioning step of the **aggregation phase**, SSI assembles tuples belonging to the same buckets in the same partitions. Each partition may contain several groups since a same bucket holds several distinct values. The first aggregation step computes partial aggregations of these partitions and returns to SSI results of the form (*Det_Enc*(group), *nDet_Enc*(partial aggregate)). A second aggregation step is required to combine these partial aggregations and deliver the final aggregation.

Correctness. Only true tuples are delivered by TDSs and they are grouped in partitions according to the bucket they belong to. Buckets are disjoint and partitions contain a small set of grouping values so that partial aggregations can be easily computed by TDSs.

Security. SSI only sees a nearly uniform distribution of *h*(bucketId) values and cannot infer any information about the true distribution of A_G attributes. Note that *h*(bucketId) plays here the same role as *Det_Enc*(bucketId) values but is cheaper to compute for TDSs.

Efficiency. TDSs do not need to materialize a large partial aggregate structure as in *S_Agg* because each partition contains tuples belonging to a small set of groups during the first phase and to a single group during the second phase. As for *C_Noise*, this property guarantees convergence of the aggregation process and maximizes the parallelism in all phases of the protocol. But contrary to *C_Noise*, this benefit does not come at the price of managing fake tuples.

Suitable queries. This solution is suitable for both kinds of queries (i.e., with small *G* like Q1, Q2 and big *G* like Q4, Q5) both in terms of efficiency (because it does not handle fake data) and feasibility (because it divides the big group into smaller ones and assigns the tasks for TDSs).

ALGORITHM 3. Histogram-based: ED_Hist (K1, K2, Q)

Input: (TDS's side): the cryptographic keys *(K1, K2)*, query Q from Querier.
Output: The final aggregation Ω_{final}.
Call distribution discovering algorithm to discover the distribution
begin *Collection phase*
Each connected TDS sends a tuple of the form $tup_e = (h(A_G), nE_{K2}(\bar{A}_G))$ *to SSI.*
// $h(A_G)$ is the mapping function applied on the A_G.
end
begin *First aggregation phase*
repeat *//(on SSI side)*
SSI groups tup_e *with the same* $h(A_G)$
TDSs connect to SSI to parallel download these groups
until *all groups in SSI have been sent to TDSs*
foreach *TDS* $\in$ *TDSs* **do** *//(on TDS side)*
repeat
Receive tup_e *from SSI*
Decrypt tup_e: $A_G \leftarrow h^{-1}(A_{Ge})$; $\bar{A}_G \leftarrow nE_{k2}^{-1}(\bar{A}_{Ge})$
Compute the aggregate values for all groups contained in $h(A_G)$: $[A_{Gi}, AGG_i]$
until *no more tuples received from SSI*
Encrypt these aggregate values: $[E_{K2}(A_{Gi}), nE_{K2}(AGG_i)]$
Send these encrypted aggregations to SSI
end
begin *Second aggregation phase*
repeat *//(on SSI side)*
SSI groups tup_e *with the same* $E_K(A_{Gi})$
TDSs connect to SSI to parallel download these groups
until *all groups in SSI have been sent to TDSs*
foreach *TDS* $\in$ *TDSs* **do** *//(on TDS side)*
repeat
Receive tup_e *from SSI*
Decrypt tup_e: $A_G \leftarrow E_{k2}^{-1}(A_{Ge})$; $AGG \leftarrow nE_{k2}^{-1}(AGG_e)$
Compute the aggregate values for only one group A_G: $[A_G, AGG]$
until *no more tuples received from SSI*
Encrypt this aggregate value: $[E_{K2}(A_G), nE_{K2}(AGG)]$
Send this encrypted aggregation to SSI
end
Filtering phase //evaluate HAVING clause
return $nE_{k1}(\Omega_{final})$ *to Querier by SSI;*

This section shows that the design space for executing complex queries with Group By is large. It presented three different alternatives for computing these queries and provided a short initial discussion about their respective correctness, security and efficiency. Chapter 6 compares in a deeper way these alternatives in terms of performance while section 4.5 analyzes the comparison of these same alternatives in terms of security. The objective is to assess whether one solution dominates the others in all situations or which parameters are the most influential in the selection of the solution best adapted to each context.

4.4 Correctness

In scenarios where TDSs are seldom connected (e.g., TDSs hosting a PCEHR), the collection phase of the querying protocol may be critical since its duration depends on the connection rate of TDSs. However, many of these scenarios can accommodate a result computed on a representative subset of the queried dataset (e.g., if Querier wants to find out the average salary of people in France with the total population of 65 millions, it is reasonable to survey only a fraction of the population). The question thus becomes *how to calibrate the dataset subset*? Larger subsets slow down the collection phase while smaller subsets diminish the accuracy and/or utility of the results. To determine if a sample population accurately portrays the actual population, we can estimate the sample size required to determine the actual mean within a given error threshold [Cochran77].

We propose to use the Cochran's sample size formula [Cochran77] to calculate the required sample size as follow:

$$spop^{est} = \frac{z^2 * \sigma^2 * \left(\frac{pop_m}{pop_m - 1}\right)}{\lambda^2 + \left(z^2 * \frac{\sigma^2}{pop_m - 1}\right)}$$

with pop_m the size of the actual population, λ the user selected error rate, z the user selected confidence level, and σ the standard deviation of the actual population. The meaning of each parameter in this formula is explained below.

The error rate λ (sometimes called the level of precision) is the range in which the true value of the population is estimated to be (e.g., if a report states that 60% of people in the sample living in Paris have salary greater than 1300 EUR/month with an error rate of ±5%, then we can conclude that between 55% and 65% of Parisian earn more than 1300 EUR/month).

The confidence level *z* is originated from the ideas of the Central Limit Theorem which states that when a population is repeatedly sampled, the average value of the attribute obtained by those samples approaches to the true population value. Moreover, the values obtained by these samples are distributed normally around the real value (i.e., some samples having a higher value and some obtaining a lower

score than the true population value). In a normal distribution, approximately 95% of the sample values are within two standard deviations of the true population value (e.g., mean).

The degree of variability σ of the dataset refers to the distribution of attributes in the population. A low standard deviation indicates that the data points tend to be very close to the expected value; a high standard deviation indicates that the data points are spread out over a large range of values. The more heterogeneous a population, the larger the sample size required to obtain a given level of precision and vice versa.

To take into account the fact that some TDS's holders may opt out of the query, let us call opt_{out} the percentage of TDSs that opt out of the survey. Then, the required sample size we need to collect in the collection phase is:

$$S = \frac{1}{1 - opt_{out}} * spop^{est}$$

Among the three parameters, λ and z are user selected but σ is data-dependent. Cochran [Cochran77] listed four ways of estimating population variances for sample size determinations: (1) take the sample in two steps, and use the results of the first step to determine how many additional responses are needed to attain an appropriate sample size based on the variance observed in the first step data; (2) use pilot study results; (3) use data from previous studies of the same or a similar population; or (4) estimate or guess the structure of the population assisted by some logical mathematical results. Usually, z = 1.96 (i.e., within two standard deviations of the mean of the actual population) is often chosen in statistics to reflect 95% confidence level. In the experiment, because σ is data-dependent, we will vary this parameter to see its impact to S. We also vary the error rate reflecting Querier's preference.

4.5 Security Analysis

To analyze the security of our proposed protocols, we use two techniques to evaluate based on the assumption of the adversary's knowledge. In the first way, with the assumption that the attacker knows the distribution of the cleartext dataset, we use the coefficient exposure to measure how much information revealed in each

protocols. Then, in the second way with stronger assumption that the attacker knows exact probability distribution of the values within each bucket, variance is used to analyze the security.

4.5.1 Coefficient Exposure

In this section, in order to quantify the confidentiality of each algorithm, we measure the information exposure of the encrypted data they reveal to SSI by using the approach proposed in [Damiani03] which introduces the concept of coefficient to assess the exposure. To illustrate, let us consider the example in Figure 7 where Figure 7a is taken from [Damiani03] and Figure 7b is the extension of [Damiani03] applied in our context. The plaintext table Accounts is encrypted in different ways corresponding to our proposed protocols. To measure the exposure, we consider the probability that an attacker can reconstruct the plaintext table (or part of the table) by using the encrypted table and his prior knowledge about global distributions of plaintext attributes.

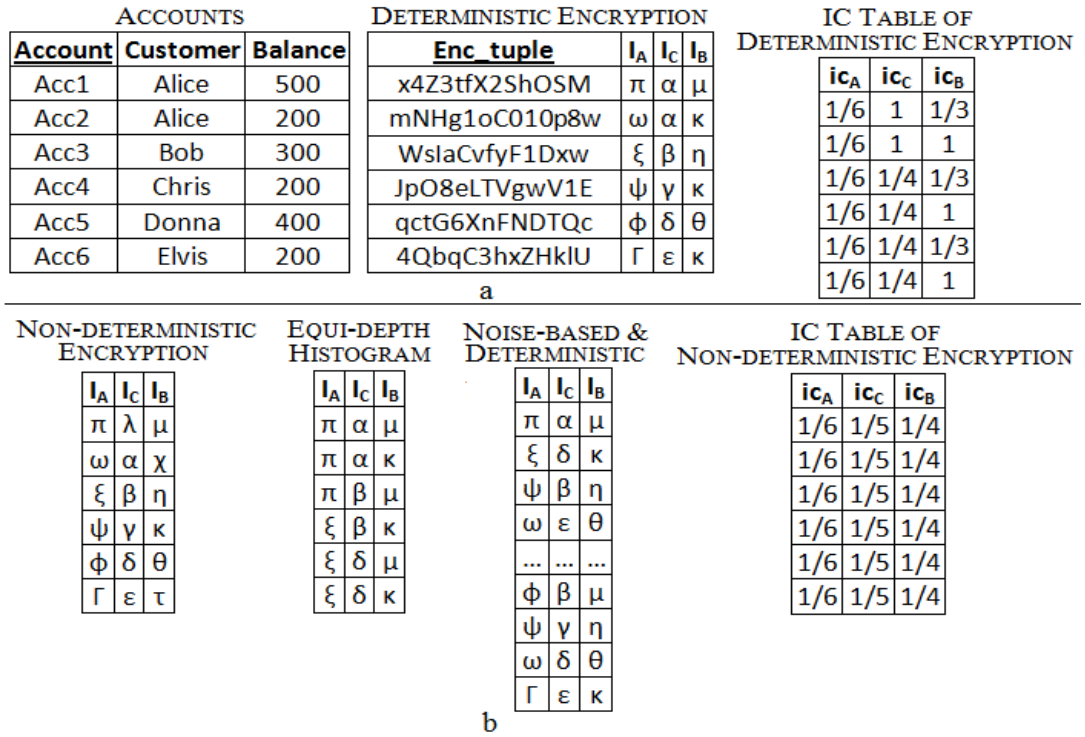


Figure 7: Encryptions and IC tables

Although the attacker does not know which encrypted column corresponds to which plaintext attribute, he can determine the actual correspondence by comparing their

cardinalities. Namely, she can determine that I_A, I_C, and I_B correspond to attributes Account, Customer, and Balance respectively. Then, the IC table (the table of the inverse of the cardinalities of the equivalence classes) is formed by calculating the probability that an encrypted value can be correctly matched to a plaintext value. For example, with *Det_Enc*, *P(α = Alice) = 1* and *P(κ = 200) = 1* since the attacker knows that the plaintexts *Alice* and *200* have the most frequent occurrences in the Accounts table (or in the global distribution) and observes that the ciphertexts *α* and *κ* have highest frequencies in the encrypted table respectively. The attacker can infer with certainty that not only *α* and *κ* represent values *Alice* and *200* (*encryption inference*) but also that the plaintext table contains a tuple associating values *Alice* and *200* (*association inference*). The probability of disclosing a specific association (e.g., *<Alice,200>*) is the product of the inverses of the cardinalities (e.g., *P(<α,κ> = <Alice,200>) = P(α = Alice)× P(κ = 200) = 1*). The *exposure coefficient Ɛ* of the whole table is estimated as the average exposure of each tuple in it.

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k} IC_{i,j}$$

Here, *n* is the number of tuples, *k* is the number of attributes, and $IC_{i,j}$ is the value in row *i* and column *j* in the IC table. Let's N_j be the number of distinct plaintext values in the global distribution of attribute in column *j* (i.e., $N_j \leq n$).

Using *nDet_Enc*, because the distribution of ciphertexts is obfuscated uniformly, the probability of guessing the true plaintext of *α* is P(*α = Alice*) = 1/5. So, $IC_{i,j} = 1/N_j$ for all i, j, and thus the exposure coefficient of *S_Agg* is:

$$\varepsilon_{S\_Agg} = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k}\frac{1}{N_j} = 1/\prod_{j=1}^{k} N_j$$

For the nearly equi-depth histogram, each hash value can correspond to multiple plaintext values. Therefore, each hash value in the equivalence class of multiplicity *m* can represent any *m* values extracted from the plaintext set, that is, there are $\binom{N_j}{m}$ different possibilities. The identification of the correspondence between hash and plaintext values requires finding all possible partitions of the plaintext values such that the sum of their occurrences is the cardinality of the hash value, equating to

solving the NP-Hard *multiple subset sum problem* [Ceselli05]. We consider two critical values of collision factor *h* (defined as the ratio G/M between the number of groups G and the number M of distinct hash values) that correspond to two extreme cases (i.e., the least and most exposure) of ε_{ED_Hist}: (1) *h* = *G*: all plaintext values collide on the same hash value and (2) *h* = 1: distinct plaintext values are mapped to distinct hash values (i.e., in this case, the nearly equi-depth histogram becomes *Det_Enc* since the same plaintext values will be mapped to the same hash value).

In the first case, the optimal coefficient exposure of histogram is:

$$\min(\varepsilon_{ED\_Hist}) = 1/\prod_{j=1}^{k} N_j$$

because $IC_{i,j} = 1/N_j$ for all i, j. For the second case, the experiment in [Ceselli05] (where they generated a number of random databases whose number of occurrences of each plaintext value followed a Zipf distribution) varies the value of *h* to see its impact to ε_{ED_Hist}. This experiment shows that the smaller the value of *h*, the bigger the ε_{ED_Hist} and ε_{ED_Hist} reaches maximum value (i.e., $\max(\varepsilon_{ED\_Hist}) \approx 0.4$) when *h* = 1.

For Noise_based algorithms, when $n_f = 0$ (i.e., no fake tuples), R_{nf}_Noise becomes *Det_Enc* and therefore it has maximum exposure in this case. If n_f is not big enough, since each TDS generates very few fake tuples, the transformed distribution cannot hide some ciphertexts with remarkable (highest or lowest) frequencies, increasing the exposure. The bigger the n_f, the lower the probability that these ciphertexts are revealed. Exceptionally, when the noise is not random (but controlled by domain cardinality of A_G), *C_Noise* has better exposure since all ciphertexts have the same frequency ($IC_{i,j} = 1/N_j$ for all i, j):

$$\varepsilon_{C\_Noise} = \frac{1}{(n_f+1)*n} \sum_{i=1}^{(n_f+1)*n} \prod_{j=1}^{k} IC_{i,j}$$

$$= \frac{1}{n_d*n} \sum_{i=1}^{n_d*n} \prod_{j=1}^{k} \frac{1}{N_j} = 1/\prod_{j=1}^{k} N_j$$

The exposure coefficient gets the highest value when no encryption is used at all and therefore all plaintexts are displayed to attacker. In this case, $IC_{i,j}$ = 1 ∀ i, j, and thus the exposure coefficient of plaintext table is (trivially)

$$\varepsilon_{P\_Text} = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k} 1 = 1$$

The information exposures among our proposed solutions are summarized in Figure 8. In conclusion, *S_Agg* is the most secure protocol. To reach the highest secure level as the *S_Agg*, other protocols must pay some high prices. Specifically, R_{nf}*_Noise* has to generate a very large amount of noise regardless of the value of *G*; *C_Noise* also incurs large noise if *G* is big; and *ED_Hist* must have a significant collision factor.

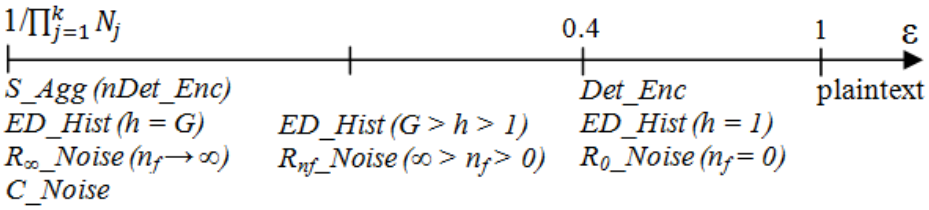


Figure 8: Information exposure among protocols

.

4.5.2 Variance

In this section, we propose a stronger assumption that the adversary (A for short) possesses more knowledge of encrypted dataset than the previous section: *A knows the entire bucketization scheme and the exact probability distribution of the values within each bucket*. For example, given that bucket B has 10 elements, we assume A knows that: 3 of them have value 85, 3 have value 87 and 4 have value 95, say. However, since the elements within each bucket are indistinguishable, this does not allow A to map values to elements with absolute certainty. Then, the A's goal is to determine the precise values of sensitive attributes of some (all) individuals (records) with high degree of confidence. Eg: What is the value of salary field for a specific tuple? [Hore04] proposes the **Variance** of the distribution of values within a bucket B

as its measure of privacy guarantee. They first define the term *Average Squared Error of Estimation (ASEE)* as follows.

Definition ASEE: Assume a random variable X_B follows the same distribution as the elements of bucket B and let P_B denote its probability distribution. For the case of a discrete (continuous) random variable, we can derive the corresponding probability mass (density) function denoted by p_B. Then, the goal of the adversary is to estimate the true value of a random element chosen from this bucket. We assume that A employs a statistical estimator for this purpose which is, itself a random variable, X'_B with probability distribution P'_B.

In other words, A guesses that the value of X'_B is x_i, with probability $p'_B(x_i)$. If there are N values in the domain of B, then we define *Average Squared Error of Estimation (ASEE)* as:

$$ASEE(X_B, X'_B) = \sum_{j=1}^{N} \sum_{i=1}^{N} p'_B(x_i) * p_B(x_j) * (x_i - x_j)^2$$

Theorem [Hore04]: ***ASEE(X, X') = Var(X) + Var(X') + (E(X) - E(X'))^2*** *where X and X' are random variables with probability mass (density) functions p and p0, respectively. Also* ***Var(X)*** *and* ***E(X)*** *denote variance and expectation of X respectively*.

Proof: interested readers refer to [Hore04] for a detail proof of this theorem.

Note that unlike coefficient exposure, the smaller value of *ASEE* implies the bigger security breach because the distance between guessed values and actual values is smaller, and vice versa. So the adversary tries to minimize *ASEE* as much as he can. From the theorem above, it is easy to see that A can minimize $ASEE(X_B, X'_B)$ in two ways: 1) by reducing $Var(X'_B)$ or 2) by reducing the absolute value of the difference $E(X_B) - E(X'_B)$. Therefore, the best estimator of the value of an element from bucket B that A can get, is the constant estimator equal to the mean of the distribution of the elements in B (i.e., $E(X_B)$). For the constant estimator X'_B, $Var(X'_B) = 0$. Also, as follows from basic sampling theory, the "mean value of the sample-means is a good estimator of the population (true) mean". Thus, A can minimize the last term in the above expression by drawing increasing number of samples or, equivalently, obtaining a large sample of plaintext values from B. However, note that the one

factor that A cannot control (irrespective of the estimator he uses) is the true variance of the bucket values, $Var(X_B)$. Therefore, even in the worst case scenario (i.e., $E(X'_B) = E(X_B)$ and $Var(X'_B) = 0$), A still cannot reduce the *ASEE* below $Var(X_B)$, which, therefore, forms the lowest bound of the accuracy achievable by A. Hence, the data owners try to bucketize data in order to maximize the variance of the distribution of values within each bucket. These two cases corresponds to the two extreme cases of nearly equi-depth histogram (when h = 1 and h = G) as analyzed below.

When h = 1 (Det_Enc), since each bucket contains only the same plaintext values, and with the assumption above about additional knowledge of adversary, he can easily infer that the expected value of X'_B equals to that of X_B: $E(X'_B) = E(X_B)$. For the variance, with h = 1, the variance of X'_B gets the minimum value $Var(X'_B) = 0$ (because variance measures how far a set of numbers is spread out, a variance of zero indicates that all the values are identical). In this case, the value of *ASEE* equals to the lowest bound $Var(X_B)$.

When h = *G*, since all plaintext values collide on the same hash value, the difference between $E(X'^2_B) - (E(X'_B))^2$ is big, leading to the big value of $Var(X'_B)$. So, the value of *ASEE* approaches highest bound.

As you can see, although the *coefficient exposure* and *average squared error of estimation* are different ways to measure privacy of equi-depth histogram depending on the adversary's knowledge, they give the same result.

Chapter 5
Implementation

In this chapter, we describe how to turn the theoretical TDS concept into a real computing infrastructure and how to solve related issues such as access control, fault tolerance, load balance in order to make the protocols feasible. We also propose the adaptive key exchange protocol, the important protocol that ensures the safety in sharing the keys among participants. Finally, we detail the prototype platform which is an instance of the architecture presented in previous chapter.

5.1 Implementation Issues

5.1.1 Making the TDS Concept Concrete

This work was partially supported by ANR grant KISS n° ANR-11-INSE-0005 with the objective to add distributed query facilities to the Personal Data Server named PlugDB[16]. PlugDB is a database engine embedded in a secure device combining the tamper-resistance of a smartcard and the storage capacity of a μSD card (see a picture of the hardware platform in Figure 14). The PlugDB embedded database engine is responsible for organizing all personal data in a relational database style, indexing the data, executing queries on it and protecting it through access control rules and encryption of the data at rest. The Yvelines district in France is currently running a large scale field experiment using PlugDB to implement a secure and portable medical-social folder improving the coordination of medical and social care at home for elderly patients.

[16] https://project.inria.fr/plugdb/

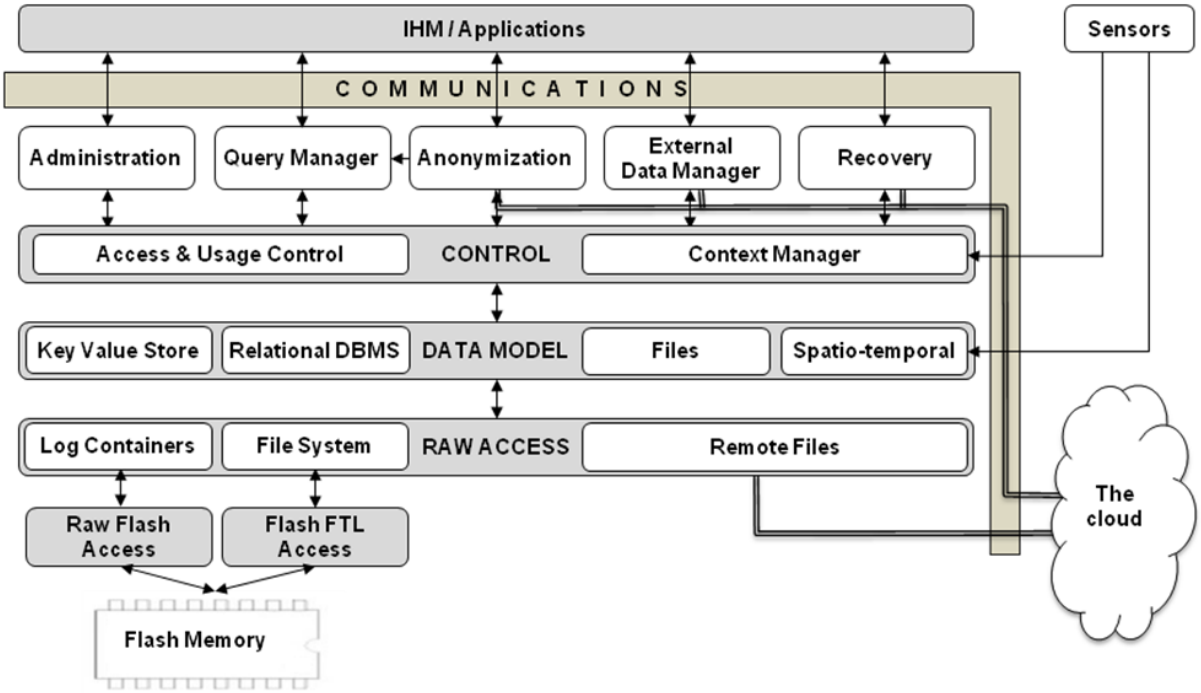


Figure 9: KISS Personal Data Server Architecture

As pictured in Figure 9, the KISS consortium extends PlugDB towards the support (1) of a wider form of personal data (spatio-temporal data, sensed data streams, documents, links to remote - encrypted - files), (2) of usage control rules, notably by integrating data provenance in the definition of the policies) and (3) of distributed facilities to execute global queries and produce anonymized releases. We refer the reader interested in a deeper description of the KISS project to [KISS12] and concentrate the next subsections to specific aspects of the KISS architecture linked to the management of distributed queries, namely how to enforce access control during query execution, how to organize the collection phase and how to organize the computation to guarantee fault tolerance and load balancing. This extension of PlugDB to distributed queries has been demonstrated in [To14b].

5.1.2 Enforcing Access Control

Contrary to statistical databases or PPDP works where the protection resides on the fact that aggregate queries or anonymized releases do not reveal any information linkable to individuals, we consider here traditional SQL queries and a traditional access control model where *subjects* (either users, roles or applications) are granted access to *objects* (either tables or views). In the fully decentralized context we are

targeting, this impacts both the definition of the access control (AC for short) and its enforcement.

AC policies can be defined and signed by trusted authorities (e.g., Ministry of Health, bank consortium, consumer association). As for the cryptographic material, such predefined policy can be either installed at burn time or be downloaded dynamically by each TDS using the key exchange protocols discussed in section 5.2. In more flexible scenarios, users may be allowed to modify the predefined AC policy to personalize it or to define it from scratch. The latter case results in a decentralized Hippocratic database [Agrawal02] in the sense that tuples belonging to a same table vertically partitioned among individuals may be ruled by different AC policies. Lastly, each individual may have the opportunity to opt-in/out of a given query. Our query execution protocol accommodates this diversity by construction, each TDS checking the querier's credentials and evaluating the AC policy locally before delivering any result (either true or dummy tuples depending on the AC outcome).

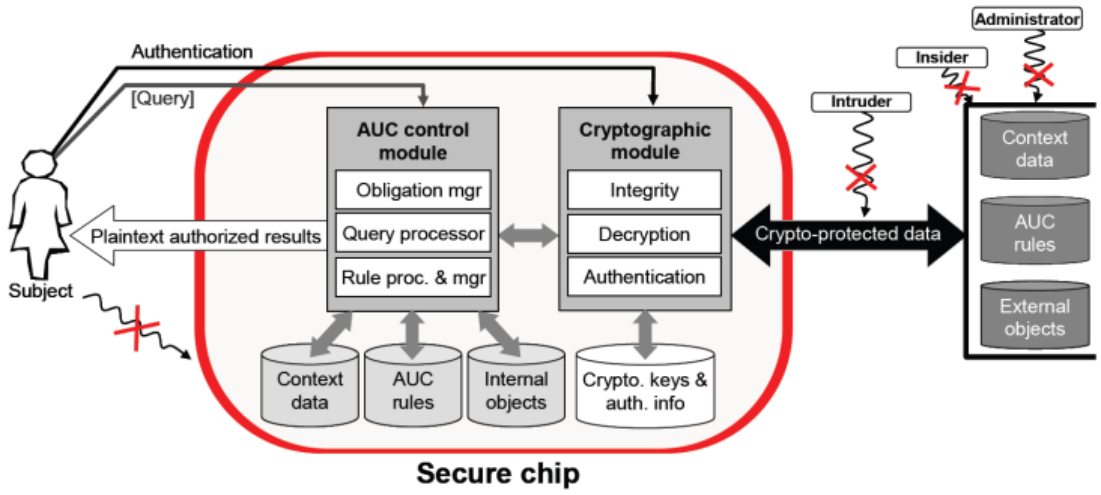


Figure 10: Functional architecture of a trusted AC system [Anciaux09]

But how can AC be safely enforced at TDS side? The querier's credentials are themselves certified by a trusted party (e.g., a public organization or a company consortium delivering certificates to professionals to testify their identity and roles). As shown in Figure 10, TDSs checks the querier's credentials and evaluates the AC policies thanks to an AC engine embedded on the secure chip, thereby protecting the control against any form of tampering. Details about the implementation of such a tamper-resistant AC module can be found in [Anciaux09].

5.1.3 Organizing the computation

Fault tolerance. In scenarios where TDSs disconnect at will, or in case of local failure, some tasks may be interrupted in the middle of their processing. To prevent data loss, SSI handles failures by re-executing the failed job on some other TDSs. SSI periodically pings the TDSs that received data in the previous steps. If SSI receives no response from a TDS after the timeout period has expired, that TDS is marked as faulty and its job is simply reassigned to another TDS. If the presumed faulty TDS finally sends its result to SSI, this result is ignored.

Load balancing. The trivial protocol in which SSI would send all collected data to a single TDS to compute the final result is meaningless in our setting because: (1) the modest storage and computing resource of a TDS would not allow it to handle such big data, (2) even if a TDS could handle that data in streaming, the computing time would not be compatible with a normal use of TDSs, considering that the primary objective of a TDS is usually *not* to participate in distributed queries (e.g., a patient plugging his TDS to update his medical record will not wait for hours in the physician's office until a distributed query is completed), (3) this TDS will become a single point of attack as in the centralized model[17].

Because of this, SSI mobilizes all connected TDSs to participate in a parallel computation of a query. The total load is distributed to available TDSs in such a way that the global execution time is minimal. In each step of the protocol, the load of the next step is smaller than that of the previous step, and therefore the number of mobilized TDSs reduces in each step (we call it the reduction factor). In the experiment section below, we use a cost model to find out the optimal reduction factor so that the execution time is smallest. The cost model also calculates the load each TDS has to incur in average.

[17] In this thesis, we make the theoretical assumption that TDSs are unbreakable. The assumption that a TDS will actually not be broken makes sense in practice thanks to the very high value of the ratio cost/benefit of an attack. However, by concentrating the computation on a single TDS, this ratio significantly decreases because a successful attack (still highly difficult to conduct) will reveal all the data.

5.2 Key Management

Our protocols rely heavily on the use of symmetric key cryptography. This section explains how these keys (k_1 for Querier-TDS communication and k_2 for inter-TDS communication) can be managed and shared in a secure way.

5.2.1 State-of-the-Art on Group Key Management

Group key management protocols can be roughly classified into three classes: centralized, decentralized, and distributed [Rafaeli03]. In centralized group key protocols, a single entity is employed to control the whole group and is responsible for distributing group keys to group members. In the decentralized approaches, a set of group managers is responsible for managing the group as opposed to a single entity. In the distributed method, group members themselves contribute to the formation of group keys and are equally responsible for the re-keying and distribution of group keys. Their analysis [Rafaeli03] made clear that there is no unique solution that can satisfy all requirements. While centralized key management schemes are easy to implement, they tend to impose an overhead on a single entity. Decentralized protocols are relatively harder to implement and raise other issues, such as interfering with the data path or imposing security hazards on the group. Moreover, distributed key management, by design, is simply not scalable. Hence it is important to understand fully the requirements of the application to select the most suitable GKE protocol. Under the computational Diffie-Hellman assumption, some works [Wu11, Bresson04] proposed group key exchange protocol suitable for low-power devices. These works achieve communication efficiency because they require only two communication rounds to establish the shared key. They also require little computing resources of participants and are thus suitable for the TDS context.

5.2.2 Overview of Key Management

There are numerous ways to share the keys between TDSs and Querier depending on which context we consider.

In the closed context, we assume that all TDSs are produced by the same provider, so the shared key k_2 can be installed into TDSs at manufacturing time. If Querier also owns a TDS, key k_1 can be installed at manufacturing time as well. Otherwise,

Querier must create a private/public key and can use another way (PKI or GKE described below) to exchange key k_1. An illustrative scenario for the closed context can be: patients and physicians in a hospital get each a TDS from the hospital, all TDSs being produced by the same manufacturer, so that the required cryptographic material is preinstalled in all TDSs before queries are executed.

In an open context, a Public Key Infrastructure (PKI) can be used so that queriers and TDSs all have a public-private key pair. When a TDS or querier registers for an application, it gets the required symmetric keys encrypted with its own public key. Since the total number of TDS manufacturers is assumed to be very small (in comparison with the total number of TDSs) and all the TDSs produced by the same producer have the same private/public key pair, the total number of private/public key pairs in the whole system is not big. Therefore, deploying a PKI in our architecture is suitable since it does not require an enormous investment in managing a very large number of private/public key pairs (i.e., proportional to the number of TDSs). PKI can be used to exchange both keys k_1 and k_2 for both Querier cases i.e. owning a secure device or not. In the case we want to exchange k_2, we can apply the above protocol for k_1 with Querier being replaced by one of the TDSs. This TDS can be chosen randomly or based on its connection time (e.g., the TDS that has the longest connection time to SSI will be chosen).

An illustrative scenario for the open context can be: TDSs are integrated in smart phones produced by different smart phone producers. Each producer has many models (e.g., iPhone 1-6 of Apple, Galaxy S1-S5 of Samsung, Xperia Z1-Z4 of Sony…) and we assume that it installs the same private/public key on each model. In total, there are about one hundred models in the current market, so the number of different private/public keys is manageable. The phone's owner can then securely take part in surveys such as: what is the volume of 4G data people living in Paris consume in one month, group by network operators (Orange, SFR…).

Another way to deliver the shared key to TDSs and Querier in the open context is to use the Group Key Exchange protocol (GKE for short) [Wu11, Amir04, Wu08] so that Querier can securely exchange the secret key to all TDSs. Some GKE protocols [Amir04] require a broadcast operation in which a participant sends part of the key to the rest. These protocols are not suitable for our architecture since TDSs communicate together indirectly through SSI. This incurs a lot of operations for SSI

to broadcast the messages (i.e., $O(n^2)$, with n is the number of participants). Other protocols [Wu08] overcome this weakness by requiring that participants form a tree structure to reduce the communication cost. Unfortunately, SSI has no knowledge in advance about TDSs thus this tree cannot be built. The work in [Wu11] proposes a protocol with two rounds of communications and only one broadcast operation. However, this protocol still has the inherent weakness of the GKE: all participants must connect during the key exchange phase. This characteristic does not fit in our architecture since TDSs are weakly connected. Finally, the Broadcast Encryption Scheme (BES) [Castelluccia05] requires that all participants have a shared secret in advance, preventing us from using it in a context where TDSs are produced by different manufacturers.

In consequence, we must propose an adaptive GKE scheme, fitting our architecture in the following section.

5.2.3 The Adaptive Key Exchange Protocol

Let p, q be two large primes satisfying $p = 2q + 1$; G_q be a subgroup of Z_p^* with the order q; g be a generator of the group G_q; H_1, H_2 be two one-way hash functions such that H_1, H_2: {0, 1}* -> Z_q^*; SID be a public session identity (note that each session is assigned a unique SID). Without loss of generality, let {Q, U_1, U_2,…, U_n} be a set of participants who want to generate a group secret key, where Q is the Querier and U_1, U_2,…, U_n are TDSs. This dynamic GKE protocol is depicted in Figure 11 and the detailed steps are described as follows.

Step 1: Each client U_i ($1 \le i \le n$) computes $r_i = H_1(Kp_i)$ and $z_i = g^{ri} \bmod p$ with Kp_i is the private key of each TDS. Then, each U_i sends (U_i, z_i) to SSI. Since all TDSs produced by the same producer share the same private/public key pair, they generate the same z_i. When this collection phase stops, SSI forwards all these (U_i, z_i) to Querier Q.

Step 2: Querier Q first selects two random values r_0, $r \in Z_q^*$ and computes $z_0 = g^{r0} \bmod p$. Upon receiving n pairs (U_i, z_i) ($1 \le i \le n$), Querier eliminates the duplicated z_i, (we assume that there remains only m pairs (U_i,z_i) with distinct z_i). Since the number of producers is very small in comparison with the number of TDSs, we have $m << n$. Q computes $x_i = z_i^{r0} \bmod p$ and $y_i = H_2(x_i || SID) \oplus r$ for i=1, 2,…, m. Finally, Q

computes the shared session key SK = $H_2(r||y_1||y_2||\ldots||y_m||SID)$ and broadcasts (Q, y_1, $y_2\ldots$, y_m, z_0, SID) to all TDSs. Since m << n, the length of the broadcast message (U_0,y_1,$y_2\ldots$,y_m,z_0, SID) is very short, saving network bandwidth.

Step 3: Upon receiving the messages (Q, y_1, $y_2\ldots$, y_m, z_0, SID), each TDS can compute $y'_i = H_2(x_i \,||\, SID')\oplus r'$ and uses r to obtain the shared key SK = $H_2(r||y_1||y_2||\ldots||y_m||SID)$. In this step, even if some TDS did not participate in the first step of the protocol, they still can get the secret group key SK because they can use their private key and the public hash function H_1 to compute the value r_i that all the TDSs belonging to the same manufacturer can compute.

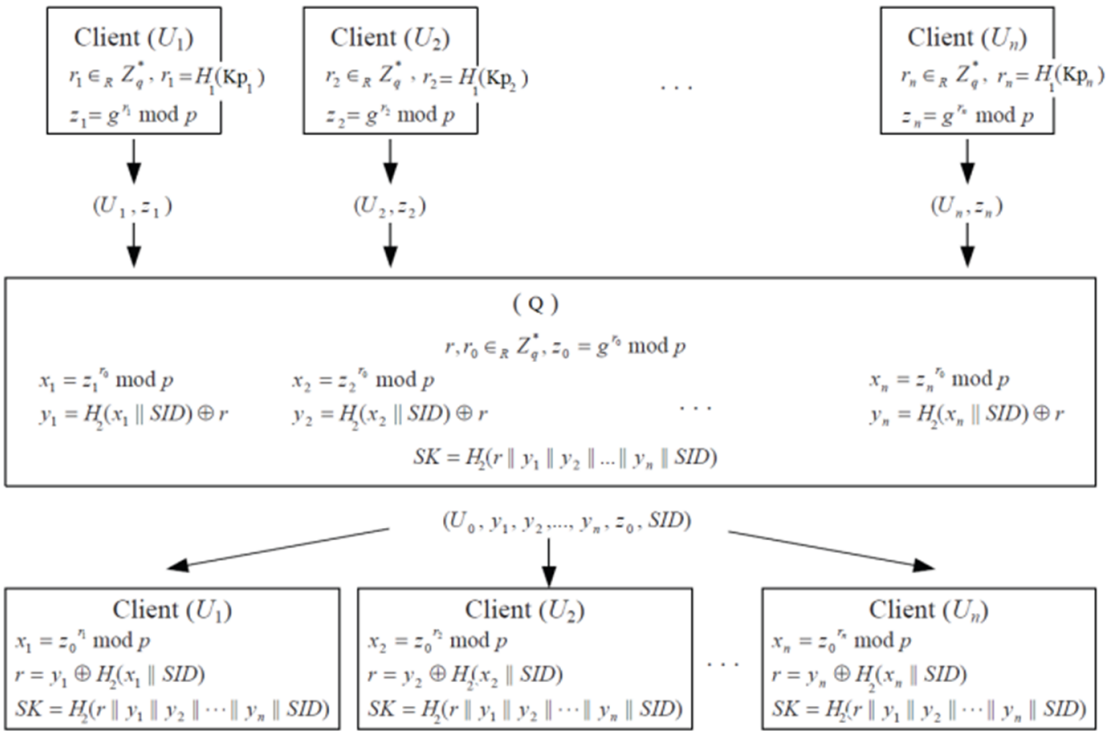


Figure 11: Adaptive Key Exchange Protocol

Note that for the security of the proposed protocol, given the Diffie-Hellman problem (see below), we make the following classical DDH and CDH assumptions, and assume there exists a secure one-way hash function.

Decision Diffie-Hellman (DDH) problem: Given $y_a = g^{x1} \bmod p$ and $y_b = g^{x2} \bmod p$ for some x1, x2 $\in Z^*_q$, the DDH problem is to distinguish two tuples (y_a, y_b, $g^{x1x2} \bmod p$) and (y_a, y_b, R $\in G_q$).

DDH assumption: There exists no probabilistic polynomial-time algorithm can solve the DDH problem with a non-negligible advantage.

Computational Diffie-Hellman (CDH) problem: Given a tuple (g, g^{x1} mod p, g^{x2} mod p) for some x1, x2 $\in Z^*_q$, the CDH problem is to compute the value g^{x1x2} mod p $\in G_q$.

CDH assumption: There exists no probabilistic polynomial-time algorithm can solve the CDH problem with a non-negligible advantage.

Hash function assumption: A secure one-way hash function H: X={0,1}* -> $Y=Z^*_q$ must satisfy following requirements:

(i) for any y $\in$ Y, it is hard to find x $\in$ X such that H(x)=y.

(ii) for any x $\in$ X, it is hard to find x' $\in$ X such that x' $\neq$ x and H(x') = H(x).

(iii) it is hard to find x, x' $\in$ X such that x' $\neq$ x and H(x)=H(x').

5.2.4 The Efficiency of the Adaptive Key Exchange Protocol

This method has two advantages in terms of asynchronous connection and performance over other GKEs in literature. First, this adaptive protocol perfectly fits our weakly connected assumption regarding the participating TDSs. Specifically, this protocol does not require that all TDSs connect at the same time to form the group, the connection of a single TDS per manufacturer being enough. The encrypted k_2 could be stored temporarily on SSI so that the offline TDS can get it as soon as it comes online and still take part in the protocol (i.e., any TDS that connects later can use its private key to compute the r_i, then *SK*, and after that can participate into the computation). Second, even if a TDS opts out of a SQL query in the collection phase, it can still contribute to the parallel computation in the aggregation phase. With a traditional distributed key exchange, any TDS disconnected during setup will require a new key exchange to take place. With our protocol, each TDS contributes to part of the shared secret key, the only requirement is that at least one TDS per manufacturer participates in step 1 to contribute to the value r_i representing this manufacturer.

In terms of performance, this protocol is not a burden because it requires only 2-round of communications as shown in Figure 11. Furthermore, the first round can be combined with the collection phase, helping reduce the protocol to only one phase.

Note that, even if SSI also possesses a TDS, it still cannot access the key shared between TDSs. As stated above, TDS code and content cannot be tampered, even by its holder. The only information that SSI in possession of a TDS can see is a stream of encrypted tuples [To14b].

Similar to PKI, adaptive GKE can be used to exchange keys k_1 and k_2 in both cases of Querier. However, although PKI and GKE are both based on the private/public keys in the open context, they differ in the way to generate the shared key. PKI is centralized and needs to trust the certification authority (which is a single point of attack) to generate the shared key. In contrast, with the adaptive GKE every TDS contributes part of the secret to generate the shared key.

5.3 Prototype: SQL/AA

In this section, we present our prototype platform and describe how we can demonstrate the proposed protocols and their scalability and parallelism, through a scenario illustrating a distributed architecture where a SSI connects to various TDSs [To14b]. To make the demonstration user-friendly and easy to follow, we use a graphical interface (Figure 13) that helps understand the overview of the system and how data flows through the system.

5.3.1 Demonstration Platform

The Hardware Platform. The demonstration platform is an instance of the architecture presented in Figure 3. A PC plays the role of the SSI, listens to connections from TDSs, manages the communication between TDSs, runs the distributed protocols, stores intermediate results, and shows encrypted data and results it receives from TDSs. A number of development boards (Figure 12) represent the TDSs and host the client application. This application can open a connection to the SSI using an Ethernet connection via a switch. These boards exhibit hardware characteristics representative of secure secure devices-like TDSs, including those provided by Gemalto (the smartcard world leader), one of our

industrial partners. This board has the following characteristics: the microcontroller is equipped with a 32 bit RISC CPU clocked at 120 MHz, a crypto-coprocessor implementing AES and SHA in hardware (encrypting or decrypting a block of 128bits costs 167 cycles), 64 KB of static RAM, 1 MB of NOR-Flash and is connected to a 1 GB external NAND-Flash and to a smartcard chip hosting the cryptographic material. Other devices used to represent the TDSs are secure devices built by the ZED company (Figure 17) that can connect to a host (e.g. a laptop connected to SSI via Ethernet) by USB port. The ZED secure devices have the same characteristics as the boards: they are equipped with a crypto coprocessor, run the same client application to receive encrypted data from the SSI, decrypt data, compute the aggregation, encrypt the result, and return the result to the SSI. Because both boards and ZED secure device are by design unobservable, they are connected to the PC through a COM port used by our demonstration to trace their behavior.

The Graphical User Interface (GUI). A GUI is used to control the system and show what information each actor can see in our system. The GUI is divided into three parts: Set of TDSs, SSI, and Querier. The first part shows the (fictional) geographic location of the TDSs. The original cleartext distribution is displayed next to it. The real distribution will be compared with the distribution of the cyphered data seen by the SSI during each protocol. The second part displays the encrypted query that the SSI receives from Querier, the encrypted data from the collection phase of each protocol and its visualization to compare the difference between protocols. The final part consists of a textbox that allows users to input any SQL query and a table to display the final cleartext result of the query.

The test platform selected was an ARM-based development board (STM32F217ZGT6[18]).

Dataset. We use a randomly-generated dataset for the demonstration. We assume that the result of the collection phase is stored in an encrypted table and all boards and ZED secure devices share the same key to encrypt/decrypt data. The cardinality of the encrypted table is one million tuples.

[18] Datasheet available at http://www.st.com/internet/mcu/product/250172.jsp (retrieved on 2012-06-15).

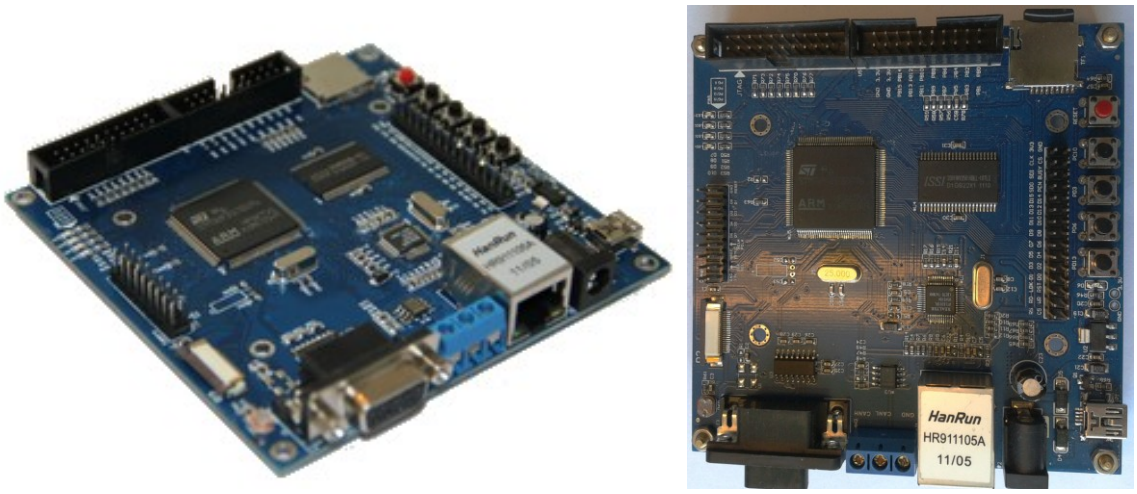


Figure 12: STM32F217 test platform

Algorithms. Our demonstration consists of three proposed protocols (i.e., S_Agg, Noise_based, ED_Hist) presented in Chapter 4, plus a Naïve protocol which simply uses deterministic encryption without any distribution obfuscation.

5.3.2 Demonstration Results

Security. Thanks to the demonstration platform, we can run the three proposed protocols, visually show the difference between their distributions, and demonstrate how they prevent frequency-based attacks. During the execution of the protocols, the platform shows what information (i.e., encrypted data) the SSI can see and demonstrate that the SSI cannot extract any meaningful information.

Performance. the platform also allows to compare the execution times of these protocols to demonstrate their performances and show their feasibility (the protocols with a small number of TDSs participating in the computation can be executed in few seconds for a dataset of one million tuples). At the end of the execution, the plaintext result is printed on the TDSs' side so that audience can compare with the SQL result executed on the plaintext table. The audience can also be invited to propose aggregate SQL queries to be tested.

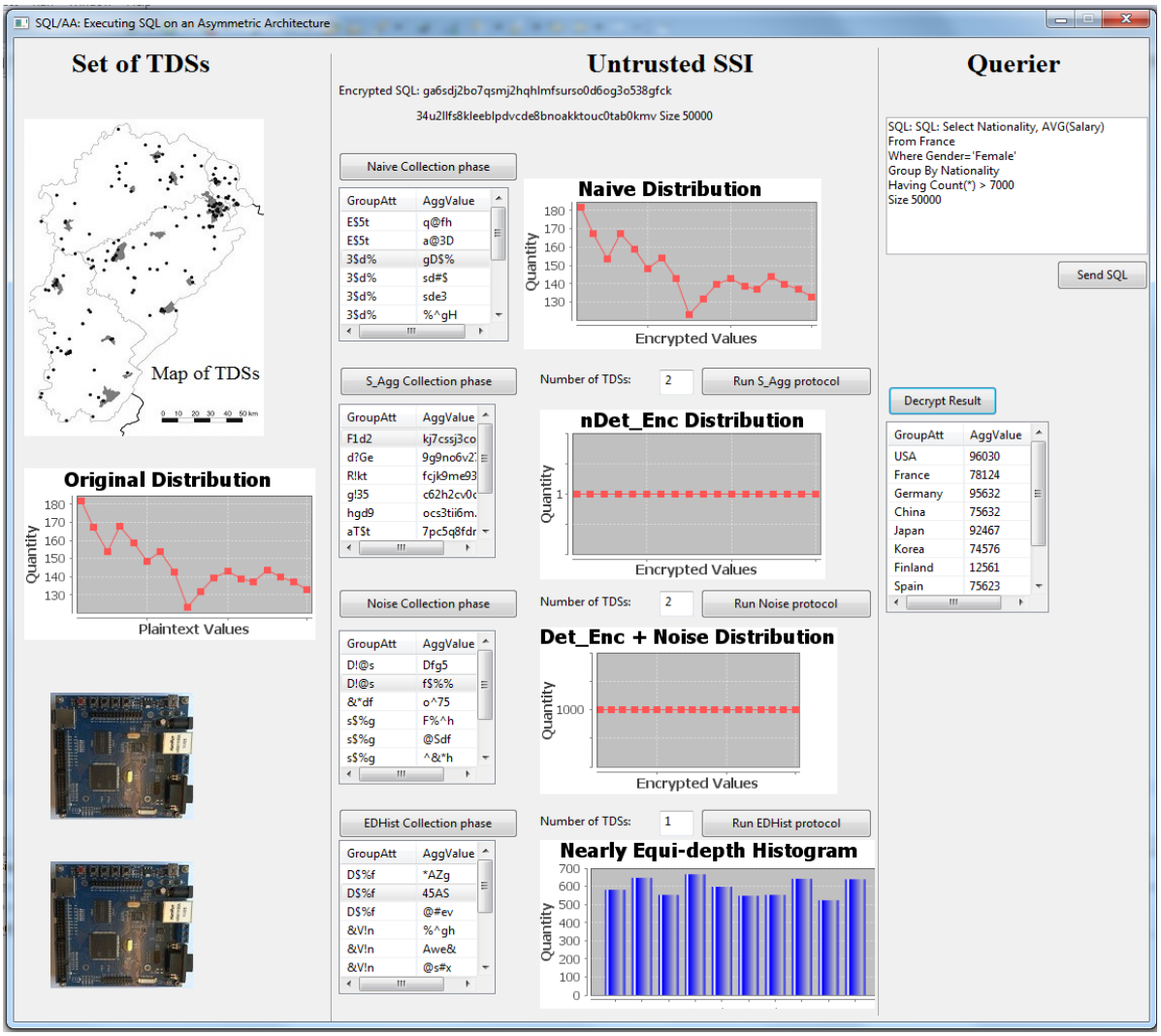


Figure 13: Demonstration graphical interface

Scalability. To show the scalability and parallelism of our system, we can vary the dataset's size and the number of TDSs used in our protocols. First, we run our protocol with one TDS, then we increase the number of TDSs to show that the execution time experimentally decreases by approximately the same value, demonstrating the scalability of the system. During the execution of the protocol, we print the encrypted intermediate result that SSI received from TDSs and show that they interleave, demonstrating parallel execution.

Chapter 6
Performance Evaluation

In this chapter, we propose a cost model to evaluate our protocols using various metrics, each representing a different aspect of the system such as execution time, data load, and resource consumption. Then we experimentally conduct unit tests on the real hardware that represents the characteristics of a secure device-like TDS. The result of these unit tests allows to calibrate the cost model and then to compare the performance between protocols. To verify the accuracy of the cost model, the final part of this chapter performs the experiment with multiple secure devices running in parallel and confronts the results with that of the cost model.

6.1 Cost Model

This section proposes an analytical cost model for the evaluation of our protocols. We calibrate this model with basic performance measurements performed on a real hardware platform (see section 6.2). We also show in section 6.3 that this model is accurate when compared to real measures on a real system composed of a set of TDSs. Thus the objective of this section is to provide an analytical model to assess the efficiency of the deployment of a TDSs based infrastructure for a given application without having to set up such a costly experiment.

The metrics of interest in this evaluation are the following.

***MaxP*$_{TDS}$**: The maximum number of TDSs *concurrently* needed in the computation. In different phases of the protocol, the optimal number of TDSs needed for the parallel computation varies and can exceed the number of connected TDSs available at that time (i.e., demanding resource is greater than available resource), reducing the

parallelism degree. Nonetheless, this value should be considered to measure the parallelism level of the protocol.

Load$_Q$: Global resource consumption for evaluating a query Q, expressed as the total size of data that all TDSs and SSI have to process. This metric reflects the scalability of the solution in terms of capacity of the system to manage a large set of queries in parallel and/or a large set of TDSs to be queried. It also provides a global view of the resource consumption (i.e., the bigger $Load_Q$, the more resource spent to process that data).

Load$_{AVG}$: Average load of all participating TDSs in the computation. While $Load_Q$ reflects the global resource consumption, this metric reflects the local resource consumption (i.e., how much load that each TDS has to incur locally in average).

Load$_{MAX}$: Maximum load of participating TDSs in the computation. Each TDS that participates in the computation incurs different load because the same TDS can participate in different steps of the protocol if connection time of that TDS is long enough. $Load_{MAX}$ reflects the possible worst case of load that a TDS can incur. This is important to measure the feasibility of the protocol. If $Load_{Max}$ is too large, maybe no TDS will ever connect for long enough.

Load$_{BL}$: Load balance among participating TDSs in the parallel computation. It is measured as the ratio of $Load_{MAX}/Load_{AVG}$. It reflects the protocol's ability to evenly divide and deliver the parallel tasks to connected TDSs.

T$_Q$: query response time, reflecting the responsiveness of the protocol. Since the time in the collection phase is application-dependent and is similar for all protocols, and since the time in the filtering phase is also similar for all protocols, T_Q focuses on the time spent on the aggregation phase, which is actually the most complex phase.

T$_{local}$: Average time that each participating TDS spends to compute the query. This metric reflects the feasibility of the solution because the longer this time, (1) the lower the probability that TDS stays connected during this time and (2) the higher the burden for an individual to accept participating in distributed queries.

s$_{RAM}$: Size of RAM required in each participating TDS for the computation.

The above metrics can be classified into: (i) Local resource consumption, reflecting the resource consumed locally in each TDS; (ii) Global resource consumption, reflecting the global resource needed for the whole system to answer a query. The weight associated to each of these metrics is context-dependent, as discussed in Section 6.2. These metrics are computed based on the following main parameters which reflect the characteristics and resources of the architecture:

- N_t total number of encrypted tuples sent to SSI by TDSs (without loss of generality, we consider in the model that each TDS produces a single tuple in the collection phase, hence N_t reflects also the number of TDSs participating in the collection phase);
- G number of groups;
- s_t size of an encrypted tuple (this size depends on the schema of the database, number of attributes needed in the query, and size of each attribute);
- T_t time spent by each TDS to process one tuple (including transfer, cryptographic and aggregation time);
- N_i^{TDS} number of TDSs that participate in the i^{th} partial aggregation phase (protocol dependent);
- α, n_{NB}, n_{ED}, reduction factors in the aggregation phase in *S_Agg, Noise_based* and *ED_Hist* respectively;
- n_f number of fake tuples per true tuple in *Noise_based* protocols;
- h average number of groups corresponding to each hash value in *ED_Hist*.

In the following sub sections, we detail the cost model for each protocol.

6.1.1 Secure Aggregation Protocol

Because the aggregation phase is iterative, the time spent in this phase is the total time for all iterative steps. In the first step of this phase, the time required to download data from SSI and return temporary result is: $t_1 = \frac{N_t}{N_1^{TDS}} * T_t$; $t_1' = G * T_t$.

Similarly, in step i of the aggregation phase, we have.

$t_i = \frac{N_{i-1}^{TDS}}{N_i^{TDS}} * G * T_t$; $t_i' = G * T_t$ $(i = 2 - n)$, with n is the total number of iterative steps in this phase.

For simplicity, we assume that the reduction factor α in every step is similar:

$\alpha = \frac{N_t/G}{N_1^{TDS}} = \frac{N_1^{TDS}}{N_2^{TDS}} = \cdots = \frac{N_{n-1}^{TDS}}{N_n^{TDS}}$.

Since $N_n^{TDS} = 1$, the number of iterative steps is $n = \left\lceil \log_\alpha \frac{N_t}{G} \right\rceil$

The computation time of *S_Agg* is:

$$T_Q^{S\_Agg} = \sum_{i=1}^{n} (t_i + t_i') = \left[(\alpha + 1) \log_\alpha \frac{N_t}{G} \right] * G * T_t$$

To find the optimal time for aggregation phase, let $f(\alpha) = (\alpha + 1)\log_\alpha(N_t/G)$.

We have: $\frac{df}{d\alpha} = \frac{\alpha * ln\alpha - (\alpha+1)}{\alpha * (ln\alpha)^2} * ln\left(\frac{N_t}{G}\right)$

Solving the equation $\frac{df}{d\alpha} = 0$ gives $\alpha \approx 3.6$.

We call $\alpha_{op} = 3.6$ the optimal reduction factor (i.e., $T_Q^{S\_Agg}$ gets the minimum value when $\alpha_{op} = 3.6$).

These other metrics are calculated as follows:

$P_{TDS}^{S\_Agg} = \sum_{i=1}^{n} N_i^{TDS} = \frac{N_t}{G} * \sum_{i=1}^{n} \alpha^{-i}$

$MaxP_{TDS}^{S\_Agg} = \frac{N_t}{\alpha G}$

$Load_Q^{S\_Agg} = (N_t + \alpha G \sum_{i=2}^{n} N_i^{TDS} + G \sum_{i=1}^{n} N_i^{TDS}) * s_t$

$= (1 + 2\sum_{i=1}^{n} \alpha^{-i}) * N_t * s_t$

$Load_{MAX}^{S\_Agg} = (n+1)\alpha G \times s_t$

$$Load_{AVG}^{S\_Agg} = \begin{cases} \frac{\left(N_t + \alpha G \sum_{i=2}^{n} N_i^{TDS}\right)}{P_{TDS}^{S\_Agg}} \times s_t, if\ P_{TDS}^{S\_Agg} < N_t \\ \frac{\left(N_t + \alpha G \sum_{i=2}^{n} N_i^{TDS}\right)}{N_t} \times s_t, if\ P_{TDS}^{S\_Agg} \geq N_t \end{cases}$$

$T_{local}^{S\_Agg} = \frac{\left(N_t + \alpha G \sum_{i=2}^{n} N_i^{TDS}\right) * T_t}{P_{TDS}^{S\_Agg}}$

6.1.2 Noise_based Protocols

Because all tuples belonging to one group may spread over multiple partitions, the aggregation phase includes two steps.

In the first step, each group contains $(n_f + 1) * N_t / G$ tuples in average, and we assume that there are n_{NB} TDSs handling tuples belonging to one group. The time required to download data from SSI and return temporary result in this step is:

$t_1 = \frac{(n_f+1)*N_t}{n_{NB}*G} * T_t$; $t_1' = T_t$;

In the second step, each TDS receives n_{NB} tuples belonging to one group to compute the final aggregation, so the time required is:

$t_2 = n_{NB} * T_t$; $t_2' = T_t$;

The computation time of R_{nf}_Noise is:

$$T_Q^{R_{nf}\_Noise} = \left(n_{NB} + \frac{(n_f+1)*N_t}{n_{NB}*G} + 2\right) * T_t$$

Apply the Cauchy's inequality, we have:

$$n_{NB} + \frac{(n_f+1)*N_t}{n_{NB}*G} \geq 2 * \sqrt{\frac{(n_f+1)*N_t}{G}}$$

The computation time of R_{nf}_Noise gets optimal value when the optimal reduction factor is: $n_{NB} = \sqrt{\frac{(n_f+1)*N_t}{G}}$.

$$P_{TDS}^{R_{nf}\_Noise} = (n_{NB}+1)*G$$

$$MaxP_{TDS}^{R_{nf}\_Noise} = n_{NB}*G$$

$$Load_Q^{R_{nf}\_Noise} = \left[(n_f+1)*N_t + 2n_{NB}*G + G\right]*s_t$$

$$Load_{MAX}^{R_{nf}\_Noise} = \left(\frac{(n_f+1)*T_{tuple}}{n_{NB}*G} + n_{NB}\right)*s_t$$

$$Load_{AVG}^{R_{nf}\_Noise} = \begin{cases} n_{NB} \times s_t, if\ P_{TDS}^{R_{nf}\_Noise} < N_t \\ \frac{((n_f+1)\times N_t + n_{NB}\times G)s_t}{N_t}, if\ P_{TDS}^{R_{nf}\_Noise} \geq N_t \end{cases}$$

$$T_{local}^{R_{nf}\_Noise} = Load_{AVG}^{R_{nf}\_Noise} \times \frac{T_t}{s_t}$$

6.1.3 Histogram-based Protocol

Let's *h* be the average number of groups corresponding to each hash value. By applying the Cauchy's inequality and the same mechanism as in R_{nf}_Noise, the optimal computation time is:

$T_{Q(op)}^{ED\_Hist} = \left(3 * \sqrt[3]{\frac{h*N_t}{G}} + h + 2\right) * T_t$ when the reduction factors in each step are: $n_{ED} = \sqrt[3]{\left(\frac{h*N_t}{G}\right)^2}$; $m_{ED} = \sqrt[3]{\frac{h*N_t}{G}}$

Then, the other metrics are based on these factors as follows:

$$P_{TDS}^{ED\_Hist} = \left(\frac{n_{ED}}{h} + m_{ED} + 1\right) * G$$

$$MaxP_{TDS}^{ED\_Hist} = max\left\{\frac{n_{ED}}{h} \times G, m_{ED} \times G\right\}$$

$$Load_{Q}^{ED\_Hist} = (N_t + 2n_{ED} * G + 2m_{ED} * G + G) * s_t$$

$$Load_{MAX}^{ED\_Hist} = \left(\frac{h \times N_t}{n_{ED} \times G} + \frac{n_{ED}}{m_{ED}} + m_{ED}\right) \times s_t$$

$$Load_{AVG}^{ED\_Hist} = \begin{cases} \frac{(N_t + n_{ED} * G + m_{ED} * G)s_t}{\left(\frac{n_{ED}}{h} + m_{ED} + 1\right)G}, if\ P_{TDS}^{ED\_Hist} < N_t \\ \frac{(N_t + n_{ED} * G + m_{ED} * G)s_t}{N_t}, if\ P_{TDS}^{ED\_Hist} \geq N_t \end{cases}$$

$$T_{local}^{ED\_Hist} = \frac{(N_t + n_{ED} * G + m_{ED} * G) * T_t}{(n_{ED}/h + m_{ED} + 1) * G}$$

Note that this is just a subset of the complete cost model which can be found in the technical report [To13].

6.2 Performance Evaluation

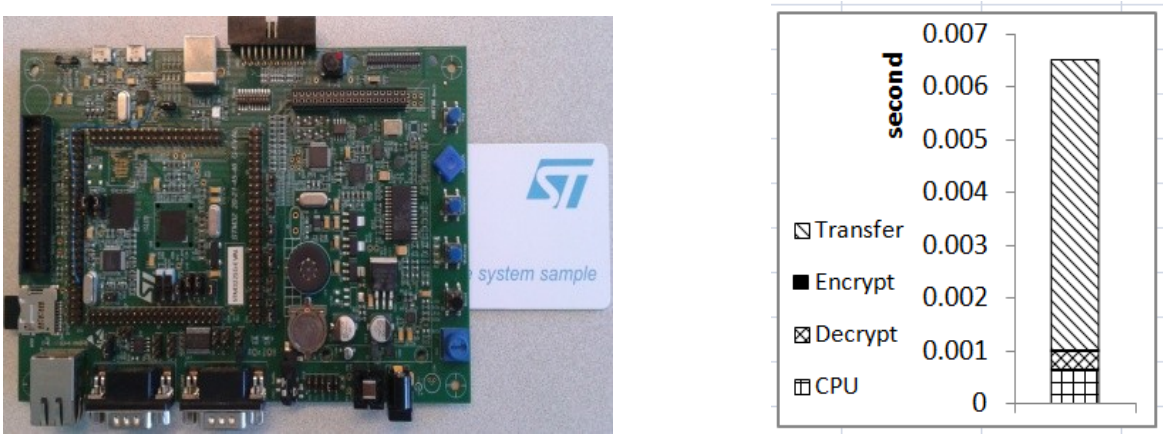
This section compares the performance among protocols using the cost model in previous section. But the result is first calibrated by using the unit test as below.

6.2.1 Unit Test

To calibrate our model, we performed unit tests on the development board presented in Figure 14a. This board exhibits hardware characteristics representative of secure secure devices-like TDSs, including those provided by Gemalto (the smartcard world leader), our industrial partner. This board has the following characteristics: the microcontroller is equipped with a 32 bit RISC CPU clocked at 120 MHz, a crypto-coprocessor implementing AES and SHA in hardware (encrypting or decrypting a block of 128bits costs 167 cycles), 64 KB of static RAM, 1 MB of NOR-Flash and is connected to a 1 GB external NAND-Flash and to a smartcard chip hosting the cryptographic material. The device can communicate with the external world through

USB full speed. The speed in theory is 12 Mbps but the real speed measured with the device is around 7.9 Mbps.

We measured on this device the performance of the main operations influencing the global cost, that is: encryption, decryption, hashing, communication and CPU time, and put these numbers as constants in the formulas. Figure 14b depicts the internal time consumption of this platform to manage partitions of 4KB. The transfer cost dominates the other costs due to the network latencies. The CPU cost is higher than cryptographic cost because (1) the cryptographic operations are done in hardware by the crypto-coprocessor and (2) TDS spends CPU time to convert the array of raw bytes (resulting from the decryption) to the number format for calculation later. Encryption time is much smaller than decryption time because only the result of the aggregation of each partition needs to be encrypted.



a) b)

Figure 14: Hardware device & its internal time consumption

Other TDSs (e.g., smart meters) may be more powerful than smart secure devices, although client-based hardware security is always synonym of low power. Anyway, as this section will make clear, the internal time consumption turns out not to be the limiting factor. Hence our choice of considering low-power TDSs in this experiment is expected to broaden our conclusions.

6.2.2 Performance Comparisons

In this study, we concentrate on the performance of Group By queries since they are the most challenging to compute. We vary the dataset size (N_t varies from 5 to 65 million), the number of groups (G varies from 1 to 10^6) as well as the number of TDSs participating in the computation as a percentage of all TDSs connected at a given time (varying from 1% to 100%). For each study, we fix two parameters and vary the others. When the parameters are fixed, N_t=10^6, G=10^3,s_t=16b, T_t=16µs, h=5 and the percentage of TDS connected is 10% of N_t. We also compute and use the optimal value for all reduction factors as well as for N_i^{TDS}. In the figures, we plot two curves for R_{nf}_*Noise* protocols, R_2_*Noise* (n_f = 2) and R_{1000}_*Noise* (n_f = 1000) to capture the impact of the ratio of fake tuples. We summarize below the main conclusions of the performance evaluation. A more detailed study is provided in a technical report [To13] and in [To14c].

In what follows, we study each of the aspects of the protocol that seem important. We draw conclusions on the use cases for each protocol in section 6.2.

Parallelism requirement (MaxP$_{TDS}$). Figure 15a presents $MaxP_{TDS}$ with varied G. Since *S_Agg* does not need too many TDSs for parallel computing, the demand of connected TDSs for computation is almost satisfied. Unlike *S_Agg*, the other solutions need a lot of TDSs for the parallel computation, and when G increases to a specific point, the available resource does not meet these demands, reducing the parallel deployment of these solutions. In Figure 15b, when G is not too big (i.e., G=1000), most of the protocols can fully deploy de parallel computation (except R_{1000}_*Noise*).

***Resource consumption* (Load$_Q$).** Figure 15c and 15d show $Load_Q$ respectively in terms of G and N_t. Not surprisingly, the total load of *Noise_based* protocols is highest because of the extra processing incurred by fake tuples. However, n_f depends only on N_t, so when G increases, the total load of *Noise_based* protocols remains constant. Other protocols generate much lower and roughly comparable loads. In general, in Figure 15d, $Load_Q$ increases steadily due to the increase of N_t.

Maximum load (Load$_{MAX}$). The maximum load of a particular TDS is illustrated in Figure 15e. In *S_Agg*, when G increases, due to the increasing size of partial aggregation, each TDS has to process bigger aggregation, resulting in the increase of $Load_{MAX}$. Also, when G increases, the number of participating TDSs decreases, so each participating TDS has to incur higher load. For others, when G increases, since N_t remains unchanged, the number of tuples in each group decreases and the number of participating TDSs increases. Consequently, each TDS processes less tuples, and thus $Load_{MAX}$ decreases. In other words, the parallel level in this case is high, reducing the maximum load that a particular TDS incurs. In Figure 15f, when N_t increases, the number of participating TDSs also increases proportionally. So, in general, the $Load_{MAX}$ remains stable except a slight increase in R_{1000}*_Noise* and *C_Noise*.

Average load (Load$_{AVG}$). Figure 15g is the average load of every participating TDS. In *S_Agg*, since the total load stays almost constant and the number of participating TDSs declines steeply when G increases, the average load goes up. In the R_{1000}*_Noise*and *C_Noise*, the high total load is constant and all available connected TDSs participate in the computation when G varies from 10^3-10^6, thus every TDSs incur the same amount of load. For the rest, $Load_{AVG}$ decreases when G increases, because there is more number of participating TDSs but the total load is almost unchanged. In Figure 15h, although *C_Noise* has higher $Load_Q$ than *S_Agg*, the number of participating TDSs in *S_Agg* is much less than that in *C_Noise*, and therefore the $Load_{AVG}$ of *C_Noise* is less than that of *S_Agg*.

Load balance (Load$_{BL}$). Figure 15i and 15j present the load balance of solutions. Because of the low parallelism, *S_Agg* is the most unbalanced protocol. R_2*_Noise* divides the load evenly among participating TDSs. *ED_Hist* has worse load balance than R_2*_Noise* since each TDS has to process a partition including *h* groups while in R_2*_Noise* a partition composes of only one group.

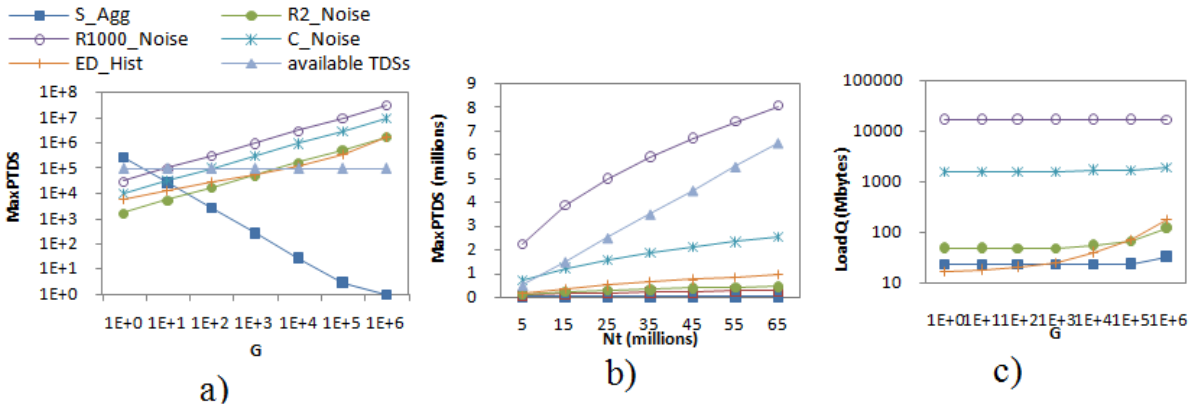
***Query response time* (T_Q).** Figure 15k shows the impact of *G* over T_Q. In all protocols but *S_Agg*, T_Q depends on the total number of tuples in each group (resp. bucket for *ED_Hist*) because all groups (resp. buckets) are processed in parallel. Hence, when *G* increases while N_t remains constant, the number of tuples in each group (resp. bucket) decreases and so does T_Q. In *S_Agg*, when *G* increases, the size of each partial aggregation increases accordingly, and so does the time to

process it and in consequence, so does T_Q. Figure 15l shows that, for *ED_Hist*, when N_t increases, the number of TDSs which can be mobilized for processing increases accordingly, leading to a minimal impact on execution time. This statement is true also for R_{nf}*_Noise* protocols with the difference that the greater number of fake tuples generates extra work which is not entirely absorbed by the increase of parallelism. For *S_Agg*, the number of iterative steps increases with N_t and so does T_Q.

Local execution time **(T_{local}).** Figure 15m and 15n plot the average execution time of every participating TDSs varying G and N_t respectively. It shows that all protocols benefit from an increase of *G* except *S_Agg*. This is due to the fact that, in *S_Agg*, less TDSs can participate in the parallel computation, and therefore each TDS has to process a higher load of bigger partial aggregations. Other protocols benefit from the fact that the computing load is shared evenly between TDSs. Figure 15n shows that all protocols but *Noise_based* protocols are insensitive to an increase of N_t again thanks to independent parallelism. The bad behavior of *Noise_based* protocols is explained by the fact that the number of fake tuples increases linearly with N_t and this increased load cannot be entirely absorbed by parallelism because the number of TDSs available for the computation is bounded in this setting by 10% of the participating TDSs.

Throughput. In general, throughput is the amount of work that a computer can do in a given period of time. Applied in our case, throughput is measured as the number of queries that our distributed system can answer in a given time period, reflecting the efficiency of our protocols (cf., Figure 15o and 15p). In Figure 15o, when G increases, the number of participating TDSs for each query increases and the execution time for each query does not reduce considerably, resulting in the reduction of throughput for all solutions. The throughput of *S_Agg*, however, increases because P_{TDS} reduces much faster than the execution time for each query when G increases. In Figure 15p, when N_t increases, the throughput remains constant for all solutions due to the proportional increase of participating TDSs. The *ED_Hist* solution has the highest throughput because it needs least participating TDSs and shortest execution time for each query. For *S_Agg*, although the response time for each query is long, the P_{TDS} is very low, resulting in high throughput. For the R_{1000}*_Noise*, since it not only demands very high number of P_{TDS} (to process fake tuples), but also responses slowly for each query, its throughput is worst.

Elasticity issues. A distributed and parallel system is said to be elastic if it can mobilize smoothly a variable part of its computing resources to meet run time requirements. Figure 15q,r,k measures the elasticity of all protocols by varying the computing resource and assessing its impact on T_Q. The computing resource is materialized here by the number of TDSs which can be mobilized to contribute to a given computation. It is expressed by a percentage of the TDSs contributing to the collection phase. Figure 15q (resp. Figure 15r, Figure 15k) considers scarce (resp. abundant, intermediate) computing resource in the sense that only 1% (resp. 100%, 10%) of the TDSs contributing to the collection phase contributes to the rest of the query computation. Comparing these figures shows that, when the resource is scarce, the parallel computation is not completely deployed, resulting in a longer time to answer the query and vice-versa. Since *S_Agg* does not depend on the number of available TDSs (but on *G* and on the memory size of TDS), its performance is not impacted by a fluctuation of the resource available. In other words, *S_Agg* has lowest elasticity.



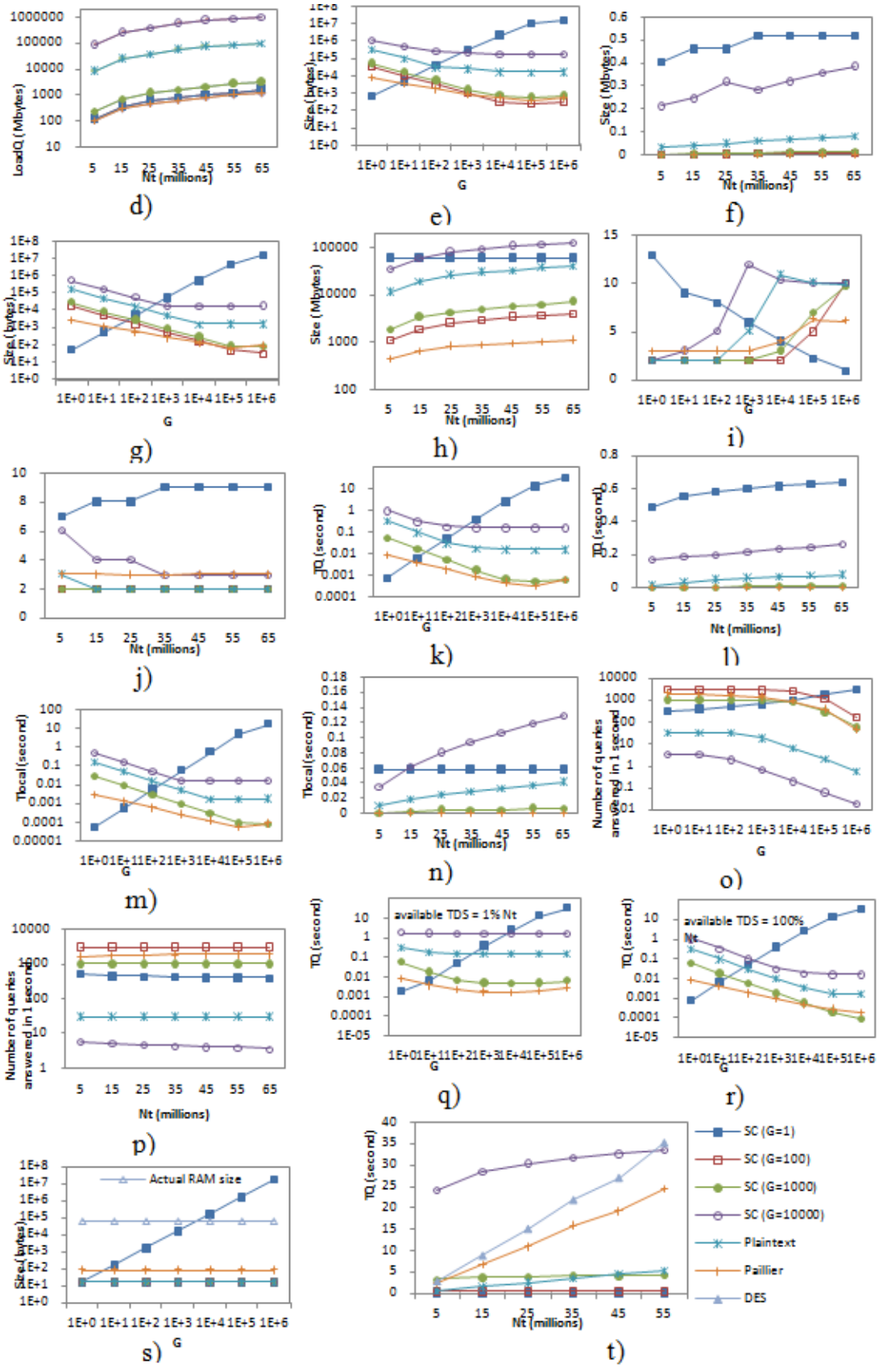


Figure 15: Performance evaluations

Memory size. Figure 15s details the memory's size required for the computation in each TDS when G is varied. Because the only factor that impacts the memory's size requirement is G but not N_t, we assess this metric by varying only G. The *Noise_based* solutions require least memory because each partition sent to TDS contains tuples belonged to only one group due to the *Det_Enc*, and thus TDSs store only one group in memory regardless of the value of G. The *ED_Hist* requires more memory because each TDS needs to process the partition having the same hash value and each hash value corresponds to multiple (i.e., *h*) groups in the first aggregation phase. The *S_Agg* needs highest memory because each TDS has to store the whole partial aggregation (which includes many groups) in the RAM. So, when G increases, the memory needed for storing the whole aggregation also increases linearly. When G is too big (i.e., G >1000), the s_{RAM} exceeds the actual RAM's size of TDS, and thus *S_Agg* is not feasible in this case[19].

6.2.3 Comparisons with State of the Art

In order to provide a baseline comparison in terms of performance (and not security), Figure 15t compares the performance of *S_Agg*, our most secure solution, with server-based solutions working on encrypted data. We consider the performance of two well-known encryption schemes, a symmetric one (i.e., DES) and a homomorphic one (i.e., Paillier [Paillier99]), as measured in [Ge07]. In DES method, each value is decrypted on the server and the computation is performed on the plaintext. Clearly this method is not a viable solution in our security model, because the database server must have access to the secret key or plaintext to answer the query, violating the security requirements. In Paillier's method, the secure modern homomorphic encryption scheme, which typically operates on a much larger (encryption) block size (say 2K bits) than single numeric data values, is used to densely pack data values in an encryption block. Then, the database server performs the computation directly on ciphertext blocks which are then passed back to a trusted agent (i.e., the Key Holder) to perform a final decryption and simple calculation of the final result. The strength of this method is due to the dense packing of values to reduce the number of modular multiplications and the minimization of the number of expensive decryption operations. We refer to the author's experimentations, which

[19]Swapping between FLASH memory and RAM is used in this case

were run on now outdated hardware[20], since both methods were implemented in C-Store[21]which was run on a Linux workstation with an AMD Athlon-64 2Ghz processor and 512 MB memory [Ge07]. We also compare its performance with C-Store using no encryption at all. We ran an AVG query varying *G* and the database size. The result (Figure 15t) shows that, with homomorphic encryption scheme (generalized Paillier), C-Store runs slightly faster than using DES for encryption due to the saving in the decryption cost during execution. It turns out that *S_Agg* outperforms DES and Paillier when the number of grouping attributes is small (i.e., G ≤ 1000) since it can exploit the parallel calculation of TDSs to speed up the computation and becomes worse after this threshold.

Although these algorithms are a little dated, the objective is simply to provide a baseline comparison, to show the effectiveness of our approach and demonstrates the strength of large-scale parallel computation even when modest hardware is available on the participant's side. Figure 15t matches this objective explicitly.

6.2.4 Trade-off between Criteria

Figure 16 summarizes and complements the experimental results described above through a qualitative comparison of our proposed protocols over all criteria of interest to perform a choice.

Each axis can be interpreted as follows. Local resource consumption axis refers to T_{local} metrics and compares the protocols in terms of feasibility, i.e., is the resource consumed by a single TDS compatible with the actual computing power of the targeted TDSs. This question is particularly relevant for low-end TDSs (e.g., smart secure devices) and of lesser interest for high-end TDSs. *S_Agg* is at the worst extremity of this axis because the final aggregation must be done by a single TDS while *ED_Hist* occupies the other extremity thanks to its capacity to evenly share the load among all TDSs. That also explains why in Load Balance axis *ED_Hist* better balances the load among TDSs than *S_Agg*. *Noise_based* protocols are in between because they also share the load evenly but at the price of managing a large number of fake tuples. Note that the relative position of *S_Agg* and *ED_Hist* is reversed in

[20] However, this hardware is still orders of magnitude superior to the secure secure devices we use.
[21] http://db.csail.mit.edu/projects/cstore/

the Global Resource Consumption and Satisfied Level of Parallel Deployment axis which refers to $Load_Q$ and $MaxP_{TDS}$ metrics and compares the scalability of the protocols in terms of number of parallel queries which can be computed and their ability of fully parallel computation, respectively. Indeed, the total number of TDSs mobilized by *S_Agg* for one single query computation is much smaller than that of *ED_Hist*. Regarding the Responsiveness axis, the relative ordering of *S_Agg* and *ED_Hist* actually differs depending on G. According to Figure 15, *S_Agg* outperforms *ED_Hist* for small G (smaller than 10) and is dominated by *ED_Hist* for larger G. Finally, Elasticity axis is a direct translation of the conclusions drawn in Section 6.2 and Confidentiality axis recalls the conclusion of Section 4.5.

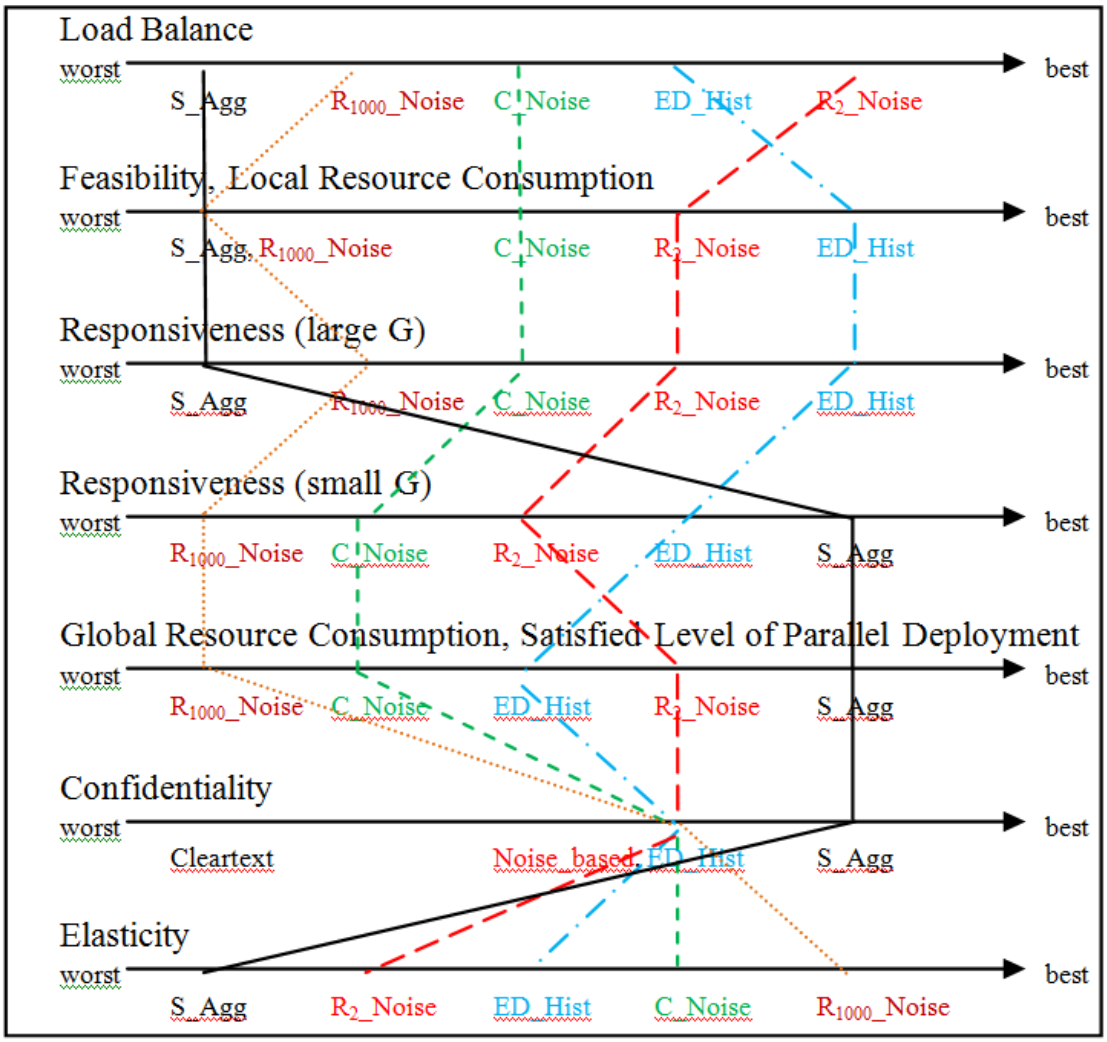


Figure 16: Comparison among solutions

This figure makes clear that *Noise_based* protocols are always dominated either by *S_Agg* or *ED_Hist* and should be avoided. However, choosing between the other two depends on the application's characteristics, and Figure 16 should be used to decide. Let us consider a first scenario where individuals manage their data (e.g., their medical folder) using a secure Personal Data Server embedded in a smart secure device-like TDS [Allard10]. In such a scenario, individuals are likely to connect their TDS seldom, for short periods of time (e.g., when visiting a doctor) and would prefer to save resource for executing their own tasks rather than being slowed down by the computation of external queries. According to Figure 16, *ED_Hist* best matches the above requirements. Conversely, let us consider a smart metering platform composed of power meter-like TDSs, connected all the time and mostly idle. In this case, TDSs' owners do not care how much resources are monopolized to compute queries and the primary concern is for the distribution company to maximize the capacity to perform global computation. *S_Agg* is more appropriate in this case. In short, *ED_Hist* and *S_Agg* are the two best solutions and the final choice depends on the weight associated to each axis for a given application.

6.3 Performance measurement on real hardware

To test the accuracy of our proposed cost models given in previous section, we compare the values taken from experiments conducted on real hardware with that of the cost models.

6.3.1 Experiment Setting

This section experimentally verifies the proposed cost models using 20 ZED secure devices[22] (Figure 18) playing the role of a pool of TDSs used during the processing phase (ie. after the collection phase has been performed). The experiment is tested on a Centrino Core 2 Duo PC with 2.4 Ghz CPU and 4 GB RAM, playing the role of SSI. The 20 ZED secure devices communicate with the PC through USB port (Figure 17). We verify our cost models on (i) Query response time (T_Q), (ii) Resource consumption ($Load_Q$), (iii) Local execution time (T_{local}) and (iv) Load balance ($Load_{BL}$)

[22] These secure secure devices are used in different universities and FabLabs in France and will be soon distributed under an open-hardware licence. In terms of hardware resources, they share many commonalities with the development board described in Section 6.2.

among secure devices. The low number of secure devices has an influence on a certain number of results, but overall we believe that our prototype demonstrates that the cost model is accurate.

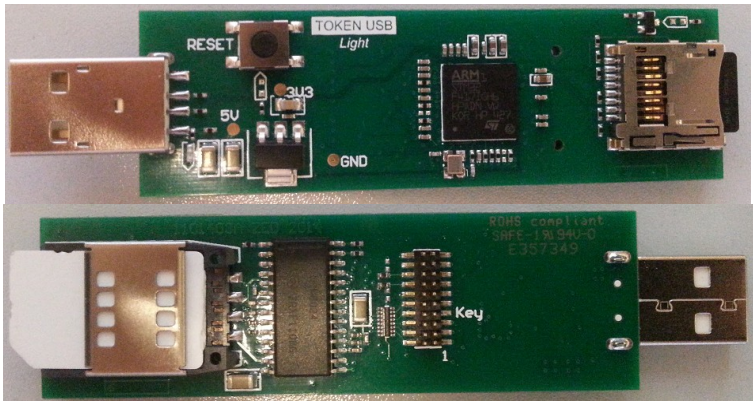


Figure 17: ZED Secure device (front & back)

The prediction accuracy is measured as the error between actual and estimated values in answering a query. Specifically, let *act* be the actual values when running on real secure devices and *est* be the estimated values when applying our proposed cost model, we adopt the following error rate definition [Tao03]:

$$Err = \frac{|est - act|}{act}$$

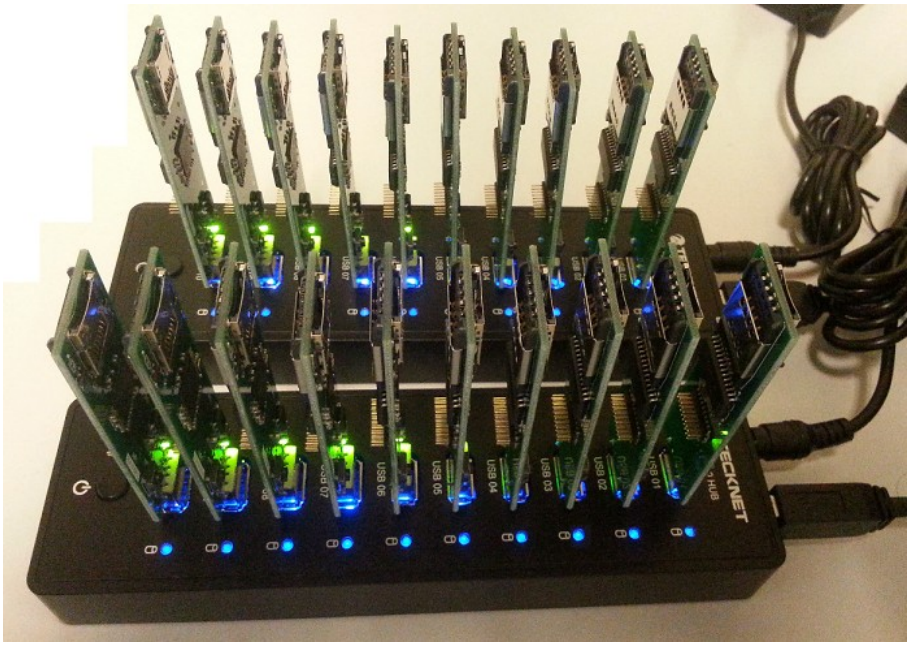


Figure 18: Twenty secure devices running parallel

Similar to the performance comparison done with the cost model in the previous section, we vary two parameters (i.e., G and N_t) to see its impact to the error rate. When N_t varies up to one million, G is fixed at 100, and when G varies from 50 to 400 groups, N_t is fixed to one million tuples.

6.3.2 Comparison

In the following figures, for each metric, the first graph represents the real absolute value measured using the 20 ZED secure devices while the second graph represents the relative error between these real values and the values predicted by the cost model. This second graph captures the accuracy of our cost model.

The first set of experiments verifies the correctness of the query response time. Figure 19a plots T_Q varying G. The Noise protocol has the longest execution time due to fake tuples, and S_Agg runs longer than ED_Hist since each secure device has to process large partial aggregation. This observation is similar to that in Figure 15k, giving a maximum estimation error under 7% in Figure 19b. When N_t varies, T_Q increases linearly in Figure 19c, similarly to Figure 15l. However, the increase rate of Figure 19c is bigger than that of Figure 15l because in the case of 20 participating secure devices, parallelism is not fully deployed due to the limited number of secure devices. On the contrary, in Figure 15l where we have many participating TDSs, the parallel computation is completely deployed, resulting in a lower increase rate when the data load increases. The maximum error is around 10% in Figure 19d.

Figures 19 e-h shows the resource consumption error rate. Similar to Figure 15c, all protocols in Figure 19e incur constant loads (except a very small increase in case of S_Agg) when G varies because the total number of tuples is fixed. This gives a very low error rate for ED_Hist and Noise protocols (around 2%) and a rather low error rate for S_Agg (less than 8%). Similarly, the variation of N_t yields the linear increase of $Load_Q$ in both Figures 15d and 19g, giving an accurate result (around 2%-3% error) in Figure 19h.

Figure 19 i-l depicts the error rate on local execution time. Except the small linear increase of S_Agg in figure 19i, Noise and ED_Hist remain constant. This contradicts the decreasing trend of Noise and ED_Hist in Figure 15m when G varies. This can be explained again by the limited number of secure devices. If the global data load

keeps unchanged, and the number of secure devices remains at twenty, each secure device processes the same amount of data in average even when G varies (except for S_Agg since the size of the aggregations depends on G). In contrast, when G increases in Figure 15m, the number of participating secure devices also increases, reducing the average connecting time for each secure device to process less load. Notice that when G increases over 1000 in Figure 15m, the T_{local} of C_Noise and R_{1000}_Noise also remains constant since the number of connecting TDSs is less than the required TDSs to fully deploy parallel computation. We believe this explanation reinforces the credibility of our cost model since this trend repeats in Figure 19i. When varying N_t, all protocols increase linearly in the experiment (Figure 19k), while they remain unchanged in the cost model (Figure 15n), except for Noise protocols. The reason of this difference is that when the total load increases while the number of secure devices remain fixed (Figure 19k), or when the number of secure devices increases but does not meet the demand for an optimal parallel computing (Noise protocols in Figure 15n), each secure device has to connect longer to process a bigger load. This is not the case for S_Agg and ED_Hist in the cost model since the increase rate of total load is less than that of connecting TDSs (in the cost model we assume that the percentage of connected TDSs is 10% of N_t).

Figure 19m displays the error rate of load balance among secure devices. Since the total load is divided evenly among twenty secure devices, the load balance remains at approximately 1 because all twenty secure devices incur nearly the same load, yielding extremely accurate prediction (with maximum error less than 2% in Figure 19n, except for S_Agg). Similarly, when N_t varies in Figure 19o, Noise and ED_Hist have better load balance than S_Agg since some secure devices in S_Agg have to process big aggregations to produce the final result. This observation conforms to the Figure 15j where S_Agg has also the most unbalanced load among protocols.

As a summary of this section, although we can measure some differences between the cost model predictions and the real measurements, the error rate remains around few percents and the trends of all graphs in Figure 19 are similar to the trends observed in Figure 15. We believe the differences arise mostly from the inability to fully deploy the parallel computation due to limited connecting TDSs in the experiments. We plan on experimenting on larger sets of secure devices in the future.

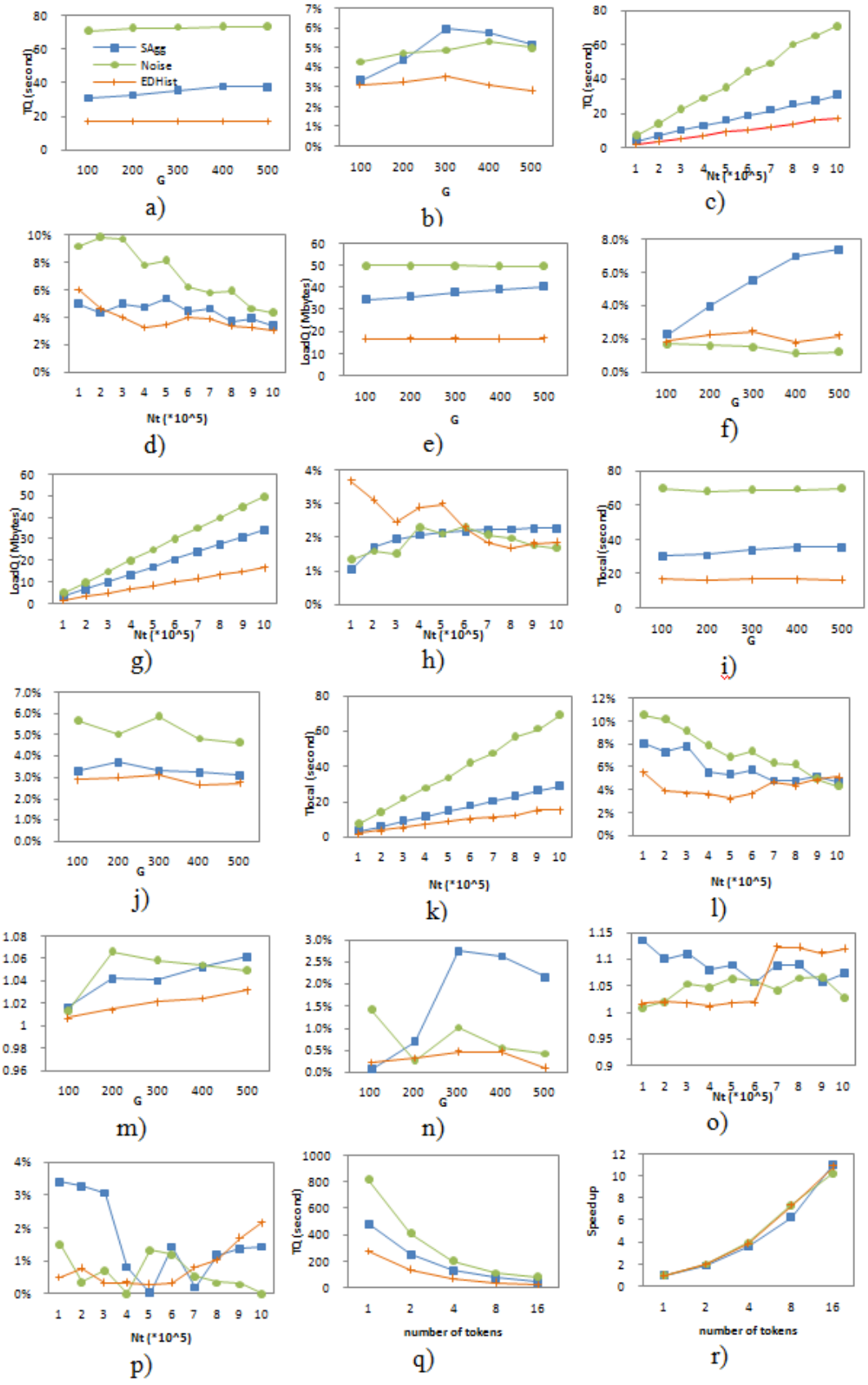


Figure 19: Performance and error rate

6.3.3 Scalability of the System

To test the ability of our system to scale up to millions of secure devices in real life applications, we measure its speedup when increasing the number of secure devices. Specifically, the speedup of our system is measured as follow:

$$S(n) = T(1)/T(n)$$

with $T(n)$ being the execution time using n secure devices.

We vary the number of secure devices to measure the execution time in Figure 19q. From that, we calculate the speedup when doubling the number of secure devices each time

In Figure 19r, the speedup approaches 12x when we use 16 secure devices. When the number of secure devices doubles, the average speedup ratios of S_Agg, Noise and ED_Hist are 1.82, 1.81 and 1.83 respectively. These speedup ratios let us expect that our system should be able to scale to millions of secure devices (given an equivalent increase in power of the SSI) in real applications with reasonable execution time and speedup. This result is not surprising considering that all protocols exhibit mainly independent parallelism.

Chapter 7
Trusted MapReduce

Relational databases were not designed to cope with the scale and agility challenges and therefore they are not suitable for big data processing that faces modern applications. Previous chapter focuses on SQL-like computation which answers the SQL queries and hence cannot tackle the more complex problems such as the key/values pair problem. This problem, which is more general than SQL queries, can be solved by MapReduce framework efficiently. With scalability, fault tolerance, ease of programming, and flexibility, MapReduce is very attractive for large-scale data processing. However, despite its merits, MapReduce does not focus on the problem of data privacy, especially when processing sensitive data on untrusted Mappers/Reducers. This chapter proposes *TrustedMR*, a trusted MapReduce system based on the *Trusted Cells* with high security assurance provided by tamper-resistant hardware, to enforce the security aspect of the MapReduce. TrustedMR pushes the security to the edges of the network where data is produced and encrypted data can be processed mostly on untrusted servers without any modification to the existing MapReduce framework. Our evaluation shows that the performance overheads of TrustedMR can easily be managed to within only few percents, compared to original MapReduce framework that handles cleartexts.

7.1 Introduction

As mentioned in previous chapters, personal data most often ends up in the Cloud for convenience and efficiency, stored within user's personal space. New companies whose business is to manage user's *personal cloud* are appearing, such as the

French company CozyCloud[23]. But more generally, companies processing sensitive, private or confidential data are looking for solutions to secure these operations, while still being able to outsource the processing. Thus an important challenge for Infrastructure as a Service (IaaS) companies is to be able to propose private and secure data management and computing to their users.

In this chapter, we focus on the MapReduce framework [Dean08]. It stands out as being the most popular solution due to its scalability, fault tolerance, ease of programming, and flexibility. With MapReduce, developers can solve various cumbersome tasks of distributed programming without the need to write complicated codes. Indeed, a developer simply writes a map and a reduce function. The system automatically distributes the workload over a cluster of commodity machines, monitors the execution, and handles failures. Current trends show that MapReduce is considered as a high-productivity alternative to traditional parallel programming paradigms for a variety of applications, ranging from enterprise computing to peta-scale scientific computing[24]. For example, power meter data can be used by the national distribution company (e.g., EDF company in France) to enable new services and products for customers. The volume of data created by energy networks is substantial, leading companies like SunEdison into big data modeling and analytics (e.g., going from one meter reading a month to smart meter readings every 15 minutes results in a huge increase data volume that must be efficiently handled). However, the *raw data* can be highly sensitive: at the 1Hz granularity provided by the French Linky power meters, most electrical appliances have a distinctive energy signature. It is thus possible to infer from the power meter data inhabitants activities [Lam07]. In consequence, raw data cannot simply be directly stored and processed on the cloud in the clear: the data must be protected. This means that if data is to be processed by the Cloud, it must be encrypted.

We consider that personal data stores are hosted by secure devices but make no additional assumption regarding the technical solution they rely on (except in the experiments section). These TDSs are deployed on the Cloud (i.e. plugged into the blade servers) in order to manage the processing of the sensitive data. The TDSs on

[23] https://cozy.io/en/
[24] http://skynet.rubyforge.org

the Cloud can either be rented by the customers of the IaaS, or could even be provided by the customers themselves, who could in particular provide some specific code to run in this secure environment.

Indeed, MapReduce was born to meet the demand of performance in processing big data, but it is still missing the function of protecting user's sensitive data from untrusted mappers/reducers. Although some state-of-the-art works have been proposed to focus on the security aspect of MapReduce, none of them aims at data privacy. They only solve the problem of integrity verification [Ruan12, Wei09] and have some weak security assumptions about untrusted servers (e.g., they require that the servers executing the Reducers must be trusted [Roy10]). Furthermore, these works often require some modifications to the original MapReduce framework to enforce the system's security (e.g., [Roy10] have to modify the original MapReduce framework to support the mandatory access control).

Based on the Trusted Cells architecture and the protocol proposed in Chapter 4, this chapter proposes a MapReduce-based system, addressing the following three important issues that every secure system must meet:

- ***Security***: How to process data using MapReduce framework without revealing sensitive information to untrusted mappers/reducers?
- ***Utility/Functionality***: How many types of operations (e.g., types of SQL queries) the proposed system can support? Can the system support key-value pair problem?
- ***Performance***: How to process large amount of encrypted data using MapReduce with small overhead, compared with performance in processing cleartext data?

To solve this problem, we consider an approach where sensitive data is produced and encrypted locally, then stored on the Cloud in the form of an encrypted data file. The processing of this data is done on the Cloud, thus we consider (as in the classical MapReduce paradigm) that this encrypted file is the input of the MapReduce task. Indeed, to ensure the data privacy, data must be obfuscated appropriately so that MapReduce framework can process encrypted data while maintaining the privacy. To ensure the utility, we transfer the encrypted data to TDSs plugged into the Cloud to decrypt and compute. Since TDSs are able to compute on

the cleartext, this approach can support any functions. To ensure the performance, especially when we have to transfer large amount of data to TDSs, we use parallel computing where each mapper/reducer splits big data into smaller ones and transfers to multiple TDSs so that they can process in parallel, reducing the transferring and computing time.

Hence, the contribution of this chapter is to propose a secure MapReduce-based system that can: (1) preserve data's privacy from untrusted mappers/reducers, (2) support unlimited types of operations, key/value pair problems and (3) have acceptable and controllable performance overhead.

7.2 Proposed Solution

Our solution is the application of *ED_Hist* protocol into the MapReduce framework. We introduce the execution phases of this framework and then explain how to apply *ED_Hist* into it.

7.2.1 MapReduce Job Execution Phases

The MapReduce programming model consists of a map(k_1; v_1) function and a reduce(k_2; list(v_2)) function. The map(k_1; v_1) function is invoked for every key-value pair <k_1; v_1> in the input data to output zero or more key-value pairs of the form <k_2; v_2>. The reduce(k_2; list(v_2)) function is invoked for every unique key k_2 and corresponding values list(v_2) in the map output. reduce(k_2; list(v_2)) outputs zero or more key-value pairs of the form <k_3; v_3>. The MapReduce programming model also allows other functions such as (i) partition(k_2), for controlling how the map output key-value pairs are partitioned among the reduce tasks, and (ii) combine(k_2; list(v_2)), for performing partial aggregation. The keys k_1, k_2, and k_3 as well as the values v_1, v_2, and v_3 can be of different and arbitrary types. The detail of map and reduce tasks is depicted in Figure 20.

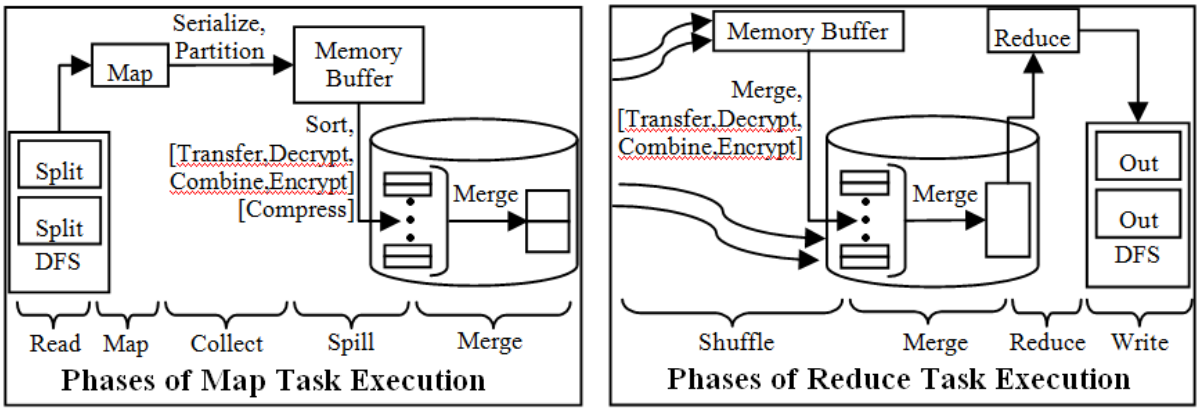


Figure 20: Detail execution of map and reduce task [Herodotou11]

In the next section, we propose a solution so that we do not need to modify this original model. We use the encryption scheme to allow the untrusted mappers/reducers participate in the computation as much as possible and transfer the necessary computations that cannot be processed on server to TDSs. These transfer and computation on TDSs happen in parallel to speed up the running time.

7.2.2 Proposed Solutions

Our proposed solution inherits the histogram-based solution, called *ED_Hist*, proposed in [To14a]. Informally speaking, to prevent the frequency-based attack on deterministic encryption (*dEnc* for short) that encrypts the same cleartexts into the same ciphertexts, and to allow untrusted server group and sort the encrypted tuples (that have the same plaintext values) into the same partitions, ED_Hist transforms the original distribution of grouping attributes, called A_G, into a *nearly equi-depth histogram* (due to the data distribution, we cannot have exact equi-depth histogram). A nearly equi-depth histogram is a decomposition of the A_G domain into buckets holding *nearly* the same number of true tuples. Each bucket is identified by a hash value giving no information about the position of the bucket elements in the domain. Figure 21.a shows an example of an original distribution and Figure 21.b is its nearly equi-depth histogram.

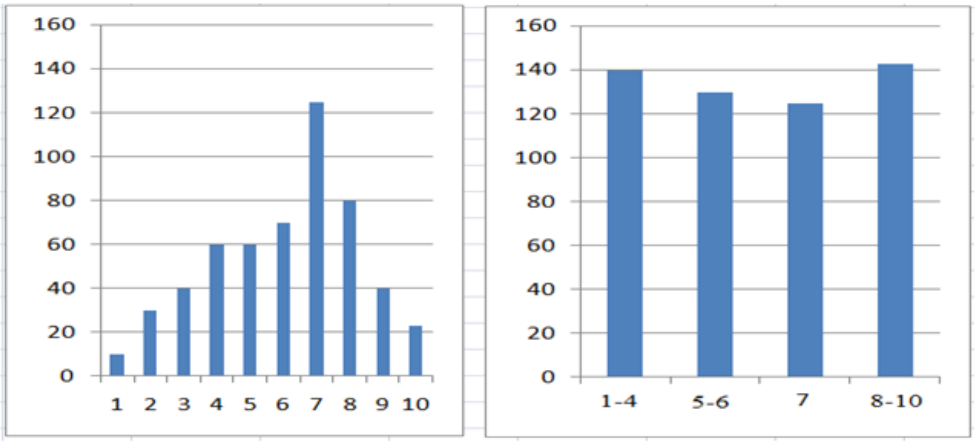


Figure 21: Example of nearly equi-depth histogram

There are three benefits in using nearly equi-depth histogram: i) allow mappers/reducers participate in the computation as much as possible (i.e., except the combine and reduce operations, all other operations can be processed in ciphertexts), without modifying the existing MapReduce framework; ii) better balance the load among mappers/reducers for skewed dataset; and iii) prevent frequency-based attack.

The protocol is divided into three tasks (see Figure 20 & 22).

Collection Task: Each TDS allocates its tuple(s) to the corresponding bucket(s) and sends to mappers/reducers tuples of the form ($\mathcal{F}(k)$, $nEnc(u)$) where $\mathcal{F}$ is the mapping function that maps the keys to corresponding buckets.

$$bucketId = \mathcal{F}(k)$$

and *nEnc* is the non-deterministic encryption that can encrypt the same cleartext into different ciphertext

Assume the cardinality of k is n, and $\mathcal{F}$ maps this domain to b buckets, then we have

$$B_1 = \mathcal{F}(k_{11}) = \mathcal{F}(k_{12}) = \ldots = \mathcal{F}(k_{1d})$$

$$B_2 = \mathcal{F}(k_{21}) = \mathcal{F}(k_{22}) = \ldots = \mathcal{F}(k_{2e})$$

$$\ldots$$

$$B_b = \mathcal{F}(k_{b1}) = \mathcal{F}(k_{b2}) = \ldots = \mathcal{F}(k_{bz})$$

From that, the average number of distinct plaintext in each bucket is:

$$h = (d + e + \ldots + z)/b = n/b$$

When this task stops, all the encrypted data sent by TDSs are stored in DFS, and are ready for processed by mappers/reducers.

Map Task: This task is divided into five phases:

1. Read: Read the input split from DFS and create the input key-value pairs: $(B_1, Enc(u_1)), (B_2, Enc(u_2)), \ldots (B_b, Enc(u_m))$.

2. Map: Execute the user-defined map function to generate the map-output data: $map(B_i; nEnc(u_i)) \rightarrow (B'_i; nEnc(v_i))$. If the map function needs process complex functions that cannot be done on encrypted data (i.e., $v_i = f(u_i)$), connections to TDSs will be established to process these encrypted data.

3. Collect: Partition and collect the intermediate (map-output) data into a buffer before spilling.

4. Spill: Sort, if the combine function is specified: parallel transfer encrypted data to TDSs to decrypt, combine, encrypt, and return to mappers, perform compression if specified, and finally write to local disk to create file spills.

5. Merge: Merge the file spills into a single map output file. Merging might be performed in multiple rounds.

Reduce Task: This task includes four phases:

1. Shuffle: Transfer the intermediate data from the mapper nodes to a reducer's node and decompress if needed. Partial merging and combining may also occur during this phase.

2. Merge: Merge the sorted fragments from the different mappers to form the input to the reduce function.

3. Reduce: Execute the user-defined reduce function to produce the final output data. Since the reduce function can be arbitrary, and therefore encrypted data cannot be executed in reducers, they must be transferred to TDSs to be decrypted, executed

the reduce function, encrypted, and returned to reducers. The difference between the output of the reduce function of traditional MapReduce with TrustedMR is that each input key represents different cleartext values, so the output key of the reduce function also represents different values: *(B'$_1$; list(nEnc(v$_1$)) -> (nEnc(k$_{11}$); nEnc(f(v$_{1i}$))), …,(nEnc(k$_{1d}$); nEnc(f(v$_{1m}$)))*.

4. Write: Compressing, if specified, and writing the final output to DFS.

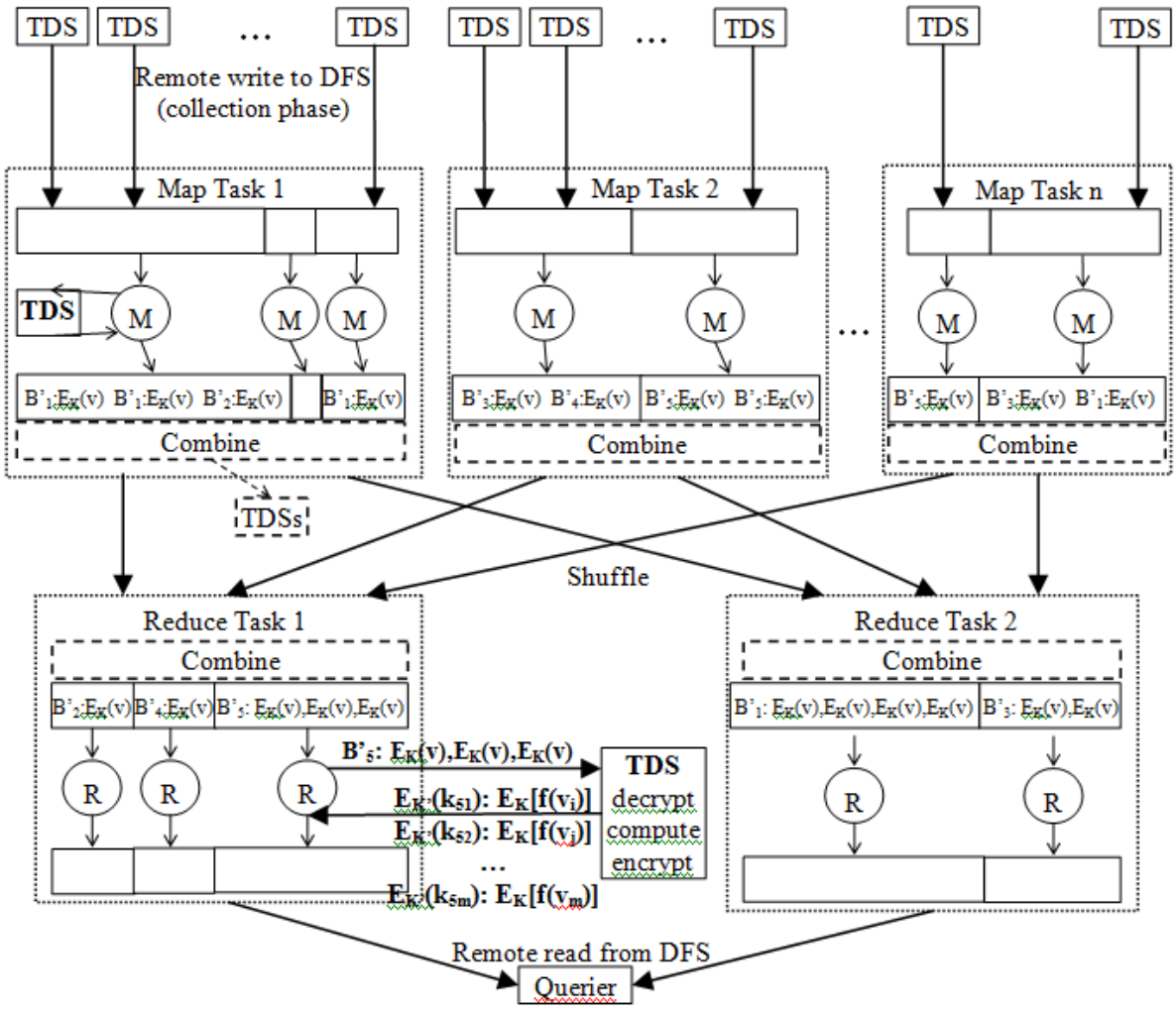


Figure 22: Trusted MapReduce execution

Among all phases in both map and reduce tasks, with the plaintext data mapped using the ED_Hist, the existing MapReduce framework can be used without being modified because each mappers/reducers can do all operations (i.e, map, partition, collect, sort, compress, merge, shuffle) on the mapped data, except the combine and reduce function. Since the combine and reduce functions must process on cleartexts,

encrypted data are transferred back to TDSs for decrypting, computing, encrypting the result and returning to mappers/reducers. To reduce the overhead of transferring large amount of data between TDSs and mappers/reducers, each mappers/reducers split the data into smaller pieces and send it in parallel to multiple TDSs. With this way, the transferring time is reduced. Below is the pseudocode for map/reduce function.

```
method Map (bucket B_i; encrypted value nEnc(u_i))
1. emit(bucket B'_i, nEnc(v_i))

method Combine (bucket B'_i; list [nEnc(v_1), nEnc(v_2),..])
1. form the partition: nEnc(v_1), nEnc(v_2),..nEnc(v_p)
2. create connection and send data to TDSs
3. in each TDS:
4.      unmap bucket: F^-1(B'_i) -> k_i1 , k_i2, .., k_in
5.      decrypt nEnc(v_i) -> v_i
6.      compute r_ij = f(v_i) having the same k_ij
7.      encrypt result r_ij -> nEnc(r_ij)
8.      map to bucket: F(k_i1) = F(k_i2) =..= F(k_in) = B'_i
9. emit (bucket B'_i; nEnc(r_ij))

method Reduce (bucket B'_i; list [nEnc(r_ij),..])
1-7. similar to Combine function from step 1 to 7
8. emit (nEnc(k_ij); nEnc(r'_ij))
```

7.2.3 How our proposed solution meets the requirements

Informally speaking, the security, utility and efficiency of the protocol are as follows (the security analysis is on Chapter 4 and we formally prove the efficiency in the next section):

Security. Since TDSs map the attributes to nearly equi-depth histogram, mappers/reducers cannot launch any frequency-based attack. What if mappers/reducers acquire a TDS with the objective to get the cryptographic material (i.e., a sort of collusion attack between mappers/reducers and a TDS)? As stated in Chapter 3, TDS code cannot be tampered, even by its holder. Whatever the information decrypted internally, the only output that a TDS can deliver is a set of encrypted tuples, which does not represent any benefit for mappers/reducers.

Utility. Since the data is processed by trusted TDSs in cleartext, our solution can support any operations.

Performance. The efficiency of the protocol is linked to the parallel computing of TDSs. Both the collection task and combine, reduce operations are run in parallel by all connected TDSs and no time-consuming task is performed by any of them. As the experiment section will clarify, each TDS manages incoming partitions in streaming because the internal time to decrypt the data and perform the computation is significantly less than the time needed to download the data. By combining the parallel computing, streaming data, and the crypto processor that can handles cryptographic operations efficiently in TDSs, our distributed model has acceptable and controllable performance overhead as pointed out in experiment.

Beside the three essential requirements above, our proposed solution meets other criteria as well: integrability and correctness.

Integrability: Because we do not need to modify the original MapReduce framework, our proposed solution can easily integrate with the existing framework. ED_Hist helps mappers/reducers run on encrypted data exactly as if they run on cleartext data without modifying the original MapReduce framework (i.e., as pointed out in previous sections, the only tasks that mappers/reducers cannot run on encrypted data are c*ombine* and *reduce*).

Correctness. Since mappers/reducers are honest-but-curious, it will strictly follow the protocol and deliver to the querier the final output. Unlike the differential privacy, mappers/reducers do not sanitize the output (to achieve the differential privacy), so the final output is correct. If a TDS goes offline in the middle of processing a partition, and therefore cannot return result as expected, mappers/reducers will resend that partition to another available TDS after waiting the response from disconnected TDS a specific interval.

7.3 Performance Evaluation

This section evaluates the performance of our solutions. We first test with the development board to see the detail time breakdown on the secure hardware (i.e., transfer, I/O, crypto, and CPU cost). Then we use the Z-secure device described below, which has the same hardware characteristic with this development board to test on the larger scale (i.e., running multiple Z-secure devices in parallel) in the real cluster. We also compare the running time on ciphertext and that on cleartext to see

how much overhead incurred. We finally increase the power of the cluster by scaling depth (i.e., increase the number of Z-secure devices plugged in each node) and scaling width (i.e., increase the number of nodes) to see the difference between the two ways of scaling.

7.3.1 Scaling with parallel computing

Experimental setup. Our experiment is conducted on a cluster of Paris Nord University with 4 nodes. Each node is equipped with 4-core 3.1 GHz Intel Xeon E31220 processor, 8GB of RAM, and 128GB of hard disk. These nodes run on Debian Wheezy 7 with unmodified Hadoop 1.0.3.

Figure 23 shows the performance overhead when processing ciphertext over cleartext. There is no difference in map time but the reduce time in ciphertext is much longer than that of cleartext. This is due to the time to connect to Z-secure device and process the encrypted data inside the Z-secure device. In this test, only one Z-secure device is plugged to each node. That creates the bottleneck for the ciphertext processing because Z-secure device is much less powerful than the node that has to wait Z-secure device to process the encrypted data. While the cleartext data is processed directly in the powerful node, the ciphertext has to be transferred to secure devices for processing. In this way, computation on ciphertext incurs three overhead in compared with the cleartext: i) time to transfer the data from node to secure device (including the connection time and I/O cost), ii) time to decrypt the data and encrypt the result, iii) the constraint on the CPU and memory size of secure device for computation inside the secure device.

To alleviate this overhead, we plug multiple secure devices to the same node and process the ciphertext in parallel in these secure devices. Figure 18 shows the 20 secure devices run in parallel and plugged to the same node. In Figure 23, when the number of secure devices plugged to each node increases, the reduce time decreases gradually and approaches that of cleartext. Specifically, when the number of secure devices increases from 1 to 20, the average speedup is 1.75. So, if we plug 32 secure devices to each node, the reduce time will be 5.49 (seconds), which gives approximate 10% longer than cleartext.

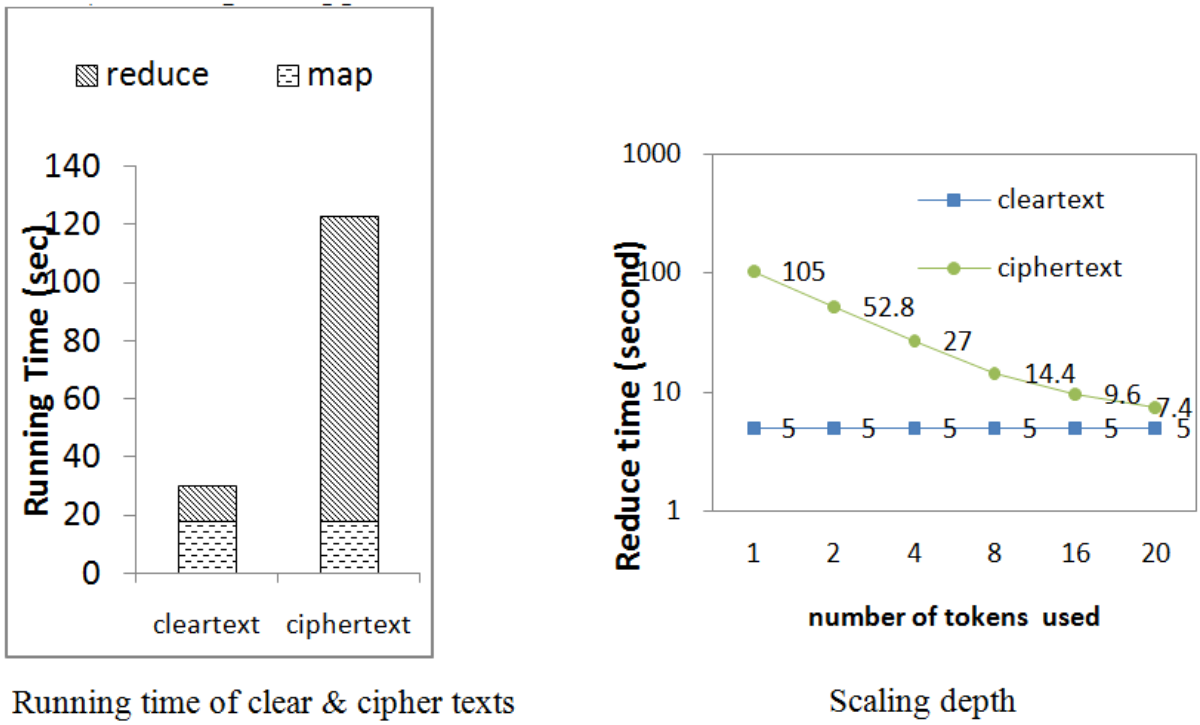


Figure 23: Running time of clear & cipher texts. Scaling depth

We observe that the overhead is controllable by increasing the number of secure devices plugged per reducer.

7.3.2 Scaling depth versus Scaling width

In traditional MapReduce, the cluster can be scaled depth by increasing number of processors per node or scaled width by increasing number of nodes. In our TrustedMR, since it depends on the secure devices for cryptographic operations, we scale depth our cluster by increasing the number secure devices (i.e., from 1 to 4) plugged to each node. We also scale width by increasing number of nodes (i.e., from 1 to 4), and then we compare the two ways of scaling. In this test, we also increase the size of the dataset (i.e., from 2 million tuples to 4 million tuples) to see how the running time varies.

In Figure 24, when we increase the number of nodes in the cluster and keep the same number of secure devices on each node, the reduce time decreases accordingly and vice versa. Also, with the same number of secure devices, plugging them to the same node or to multiple nodes gives almost no difference in term of running time (e.g., the reduce time of 4 nodes with each node having only 1 secure

device is only few percent difference from that of 1 node having 4 secure devices plugged). Furthermore, the average speedup of scaling width is 1.74 which is only 2% different from that of scaling depth (i.e., 1.71). In conclusion, scaling depth yields nearly the same performance as scaling width. The only factor that affects the overall performance of the cluster is the total number of secure devices plugged to this cluster, no matter how they are distributed to each node.

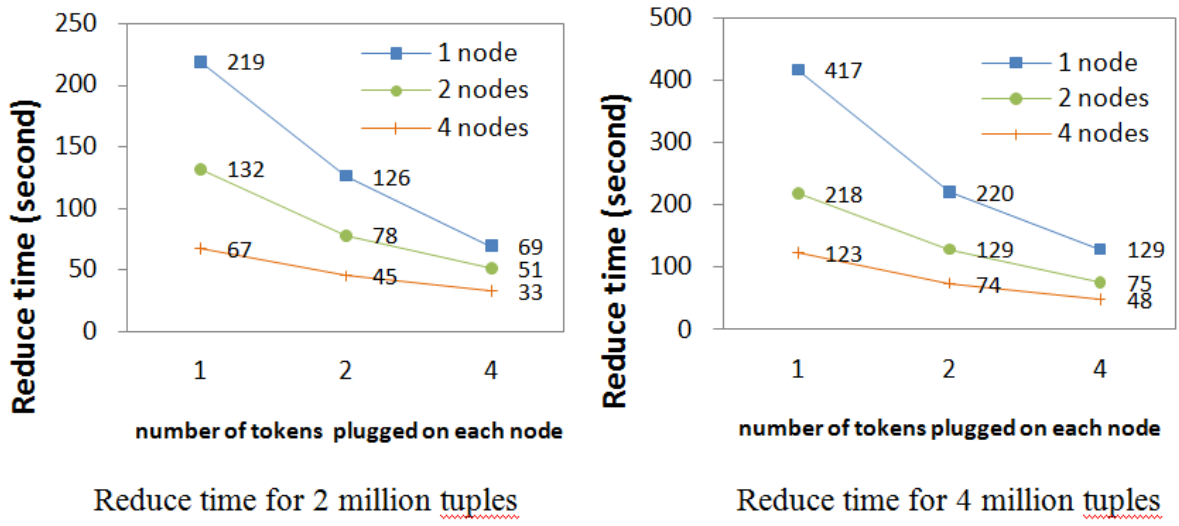


Figure 24: Reduce time for 2 millions & 4 millions tuples

7.4 Conclusion

In this chapter, we have proposed a new approach to deal with the problem of processing big data using MapReduce while maintaining privacy guarantees. Our approach draws its novelty from the fact that (private) user data remains under the control of its owner, in a Trusted Data Server. As secure hardware become available at any client device, such highly secure and decentralized architectures can no longer be ignored. The security is pushed to the edge of the network where data is produced, avoiding inherent weakness of the centralized database (single point of attack, low cost/benefit ratio). The existing MapReduce framework keeps unchanged and the types of supported operations are general. We study their efficiency in terms of running time using real secure hardware. The results show the performance overhead is acceptable (i.e., can be controlled to few percents when number of secure devices plugged to each node is big enough).

Chapter 8
Conclusion and Future Work

The massive amount of personal data is generated at a tremendous pace. Citizens have no way to opt-out because governments or companies that regulate our daily life require them. Administrations and companies deliver an increasing amount of personal data in electronic form, which often ends up in central servers at the user convenience. Although data centralization has unquestionable benefits in terms of resiliency, availability and even consistency of security policies, they must be weighted carefully against the privacy risks.

Decentralized architectures, devised to help individuals better protect their privacy, hinder global treatments and queries, impeding the development of services of great interest. This thesis is a first attempt to fill this gap. It capitalizes on secure hardware advances promising soon the presence of a Trusted Execution Environment at low cost in any client device (trackers, smart meters, sensors, cell phones and other personal devices).

The approach promoted in this thesis is part of the KISS Personal Data Server Architecture. As described in previous sections, it outlines an individual-centric architecture whose aim is to enable secure personal data server and at the same time provide control over one's data with tangible enforcement guarantees.

This chapter concludes the thesis. We synthesize the work conducted, and close the manuscript by opening exciting research perspectives.

8.1 Synthesis

We have proposed new query execution protocols to compute general SQL queries (without multi-TDS joins) while maintaining strong privacy guarantees. The objective

was not to find the most efficient solution for a specific problem but rather to perform a first exploration of the design space. By applying a variety of encryption scheme, we proposed three very different protocols and compared them according to different axes. The encouraging conclusion is that a good performance/security trade-off can be found in many situations.

As stated in Chapter 1, we address in this thesis the problem of implementing privacy-preserving SQL execution on asymmetric architecture based on distributed secure device with three main objectives (i.e., Decentralization, Security, and Generality). We summarize below how we successfully satisfy these objectives.

The approaches proposed in this thesis are based on a *Trusted Data Server* that embeds a software suite, allowing it to provide a full-fledged database engine, while enforcing the strong privacy guarantee. By using this secure device, each individual can autonomously manage his own data, and under his control, without the need of a trusted central server. In other words, the Decentralization objective is satisfied.

To prove the Generality of the system, we built a cost model to evaluate the performance of each protocol. The unit test was conducted on a development device and its result was calibrated into the cost model to produce the measurement in the large scale. We found that the proposed protocols can scale up to nation-wide contexts, proving the Generality of the system. To verify the accuracy of the cost model, we performed the experiment on multiple secure ZED secure devices running in parallel, and compared this result with that of the cost model to get the error rate. These experiments have showed the accuracy of the proposed cost model, reinforcing our proof of Generality.

In order to evaluate the Security objective, we used the two concepts of coefficient exposure and variance depending on the assumption of attacker's knowledge about dataset. In spite of the difference in the ways to measure the security, these two methods give the same conclusion: the more information exposed to supporting server to allow him to more actively participate into the computation (and thus the less security level), the higher performance we gain, and vice versa.

Finally, we have shown that these protocols can be integrated in concrete software and hardware platforms, thus providing a comprehensive solution to the problem tackled in this thesis.

8.2 Perspectives

We expect that this work will pave the way for the definition of future fully decentralized privacy-preserving querying protocols. 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.

Support Multi-TDS Joins

Privacy-preserving joins referring to Information integration across databases owned by multiple entities is important in many applications. It considers the problem of how entities compute arbitrary joins function using their data in a secure way such that no information – other than the join query results – is revealed.

While our protocols already support a wide range of queries including joins where two joining relations are inside the same TDSs, additional effort is required to support joins between several TDSs. It is more complex and time-consuming to execute these join query on distributed databases where two tables participating in a join query are stored at different TDSs. In future work, we plan on tackling the problem of joins between several or more TDSs, to support social network type queries (e.g. how many users have at least 10 friends that like "literature"). Such queries convey obvious privacy problems, but also add some extra degree of trust, due to the fact that there may exist a trust network inside the social network itself. Note that secure join protocols could be devised based on recent work on efficient secure intersect algorithms using smart cards [Fischlin11].

One straightforward solution to perform privacy-preserving multi-TDS joins is to rely on the SSI to whom all parties submit their encrypted inputs using deterministic encryption. The SSI then computes the join directly on the encrypted data and returns the results. This approach is in general easy to implement and efficient. Yet deterministic encryption is too vulnerable due to frequency-based attacks to be ubiquitously accepted by all TDSs.

Another approach is based on the secure multi-party computation problem where parties collectively perform a computation over their data. Each TDS sends encrypted data indirectly to other TDSs through SSI. However, the communication

complexity of this approach is normally too high for them to be practical, especially when we consider the unavailability constraint of TDSs in our context.

A natural question to ask is whether there exist solutions that strike a balance between the level of required trust on SSI and performance.

Extend the Thread Model

Another important research direction is to extend the threat model to (a small number of) compromised TDSs. In this thesis, the important assumption is that the microcontroller inside TDS is tamper resistant. However, in reality, TDS still can be compromised, though such attack is highly improbable due to its cost and complexity. So we must think of a solution to deal with this situation when TDSs can be compromised. In previous work, [Allard14] have shown that it is possible to convert adapt secure protocols where TDS are unbreakable to secure protocols where a small subset are corrupted.

Solutions can be devised either by clustering the keys so that breaking a TDS allows only to decrypt a random subset of the data or by providing detection mechanisms so that a compromised TDS is quickly blacklisted.

The first step of this solution is to detect which TDSs are compromised. Then, the second step is to propose the new key exchange protocol that revokes the shared key to these broken TDSs, generates the new one, and deliver to TDSs, excluding the compromised ones.

Conduct Performance Study on Large Scale Platforms

Although the cost model is verified accurately using the real secure hardware, the experiment conducted in this thesis is on quite small scale. A further study, therefore, will perform the experiment on a very large scale, and put our protocols into practice.

The on-going deployment of very large TDS platforms (e.g., the Linky power meters installed by EDF in France or the growing interest for PCEHR hosted in secure secure devices) would enable us to perform the experiment on the larger scale.

This large scale platform also provides a strong motivation to investigate two future works mentioned above.

Support MapReduce with other Architecture

The MapReduce architecture mentioned in chapter 7 in this thesis consider that map and reduce functions are executed in the untrusted Cloud with the help of TDSs plugged to mappers/reducers. In this case, TDSs partly participate in the computation. A future study will consider the case in which the map and reduce functions will be executed entirely inside the TDSs. To do this, we have to extend the PlugDB engine so that it can support the MapReduce framework. In this case, the challenge is to assert (1) that these functions are safe (since adversary can inject the malicious code inside these functions) and (2) that each TDS really executes these functions (but not others).

Bibliography

[Abadi03] Daniel J. Abadi, Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan B. Zdonik. 2003. Aurora: a new model and architecture for data stream management. VLDB Journal. 12, 2, 120-139.

[Agrawal02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2002. Hippocratic databases. In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02). Hong Kong, 143-154.

[Agrawal04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order-preserving encryption for numeric data. In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04). Paris, 563-574.

[Allard10] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Yanli Guo, Lionel Le Folgoc, Benjamin Nguyen, Philippe Pucheral, Indrajit Ray, Indrakshi Ray and Shaoyi Yin. 2010. Secure Personal Data Servers: a Vision Paper. In Proceedings of the 36th International Conference on Very Large Data Bases (VLDB'10). Singapore, 25-35.

[Allard14] Tristan Allard, Benjamin Nguyen, and Philippe Pucheral. 2014. MetaP: Revisiting Privacy-Preserving Data Publishing using Secure Devices. Distributed and Parallel Databases. 32, 2, 191-244.

[Alzaid08] Hani Alzaid, Ernest Foo, and Juan G. Nieto. 2008. Secure Data Aggregation inWireless Sensor Networks: A Survey. In Proceedings of the 6th Australasian Information Security Conference (AISC'08). 93-105.

[Amanatidis07] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. 2007. Provably-secure schemes for basic query support in outsourced databases. In DBSec. Lecture Notes in Computer Science, volume 4602, Springer. 14-30.

[Amir04] Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, and Gene Tsudik. 2004. On the performance of group key agreement protocols. ACM Transactions on Information and System Security (TISSEC). 7, 3, 457-488.

[Anciaux09] Nicolas Anciaux, Luc Bouganim, and Philippe Pucheral. 2009. Hardware Approach for Trusted Access and Usage Control. Handbook of research on Secure Multimedia Distribution (Chapter A). IGI Global.

[Anciaux13] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral and Iulian Sandu-Popa. 2013. Trusted Cells: A Sea Change for Personal Data Services. In CIDR. Asilomar, USA.

[Arasu14] Arvind Arasu & Raghav Kaushik: Oblivious Query Processing. ICDT 2014

[Bajaj11] Sumeet Bajaj, Radu Sion: TrustedDB: a trusted hardware based database with privacy and data confidentiality. SIGMOD Conference 2011: 205-216

[Bellare07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. 2007. Deterministic and efficiently searchable encryption. In CRYPTO. Lecture Notes in Computer Science, volume 4622. 535–552.

[Blass12a] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, Melek Önen: PRISM-Privacy-Preserving Search in MapReduce. In PETS, pp 180-200, 2012.

[Blass12b] Erik-Oliver Blass, Guevara Noubir, Triet Vo Huu: EPiC: Efficient Privacy-Preserving Counting for MapReduce. In IACR Cryptology ePrint Archive (2012) 452.

[Boldyreva09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-Preserving Symmetric Encryption. EUROCRYPT, pp 224-241, (2009).

[Boneh04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano. Public key encryption with keyword search. In Advances in Cryptology – EUROCRYPT '04, LNCS vol. 3027, pp. 506–522, 2004

[Bresson04] Emmanuel Bresson, Olivier Chevassut, Abdelilah Essiari, and David Pointcheval. 2004. Mutual authentication and group key agreement for low-power mobile devices. Computer Communications. 27, 17, 1730-1737.

[Castelluccia05] Claude Castelluccia, Einar Mykletun, and Gene Tsudik. 2005. Efficient Aggregation of Encrypted Data in Wireless Sensor Networks. In Mobiquitous. 109-117.

[Ceselli05] Alberto Ceselli, Ernesto Damiani, Sabrina De Capitani di Vimercati, Sushil Jajodia, Stefano Paraboschi, Pierangela Samarati. Modeling and assessing inference exposure in encrypted databases. ACM TISSEC, vol 8(1), pp. 119-152, (2005)

[Cochran77] William Gemmell Cochran. 1977. Sampling Techniques. John Wiley, 3rd edition.

[Cormode06] Graham Cormode, Flip Korn, S. Muthukrishnan, Divesh Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In PODS, pages 263–272, 2006.

[Damiani03] Ernesto Damiani, Sabrina De Capitani di Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. 2003. Balancing confidentiality and efficiency in untrusted relational DBMSs. In ACM CCS. 93-102.

[Dean08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM, 51(1):107–113, 2008.

[eMarketer12] Email Marketing Benchmarks: Key Data, Trends and Metrics. eMarketer, 2012.

[Fayyoumi10] Ebaa Fayyoumi and B. John Oommen. 2010. A survey on statistical disclosure control and micro-aggregation techniques for secure statistical databases. Software: Practice and Experience. 40, 12, 1161-1188.

[Fischlin11] Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, Ivan Visconti. Secure set intersection with untrusted hardware secure devices. In CT-RSA, (2011)

[Fung10] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. 2010. Privacy-Preserving Data Publishing: A survey of Recent Developments. ACM Computing Surveys. 42, 4, 1-53.

[Ge07] Tingjian Ge, and Stan Zdonik. 2007. Answering aggregation queries in a secure system model. In VLDB. Vienna, 519–530.

[Gentry09] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In STOC. Maryland. 169-178.

[Goldwasser84] Shafi Goldwasser and Silvio Micali, Probabilistic Encryption, Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984

[Greenwald96] Michael Greenwald. Practical algorithms for self scaling histograms or better than average data collection. Perform. Eval., 27/28(4):19–40, 1996

[Guha01] Sudipto Guha, Nick Koudas, Kyuseok Shim. Data-streams and histograms. In STOC, pages 471–475, 2001.

[Hacigumus02] Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over encrypted data in database service provider model. In ACM SIGMOD. Wisconsin, 216-227.

[Hacigumus04] Hakan Hacigümüs, Balakrishna R. Iyer, and Sharad Mehrotra. 2004. Efficient execution of aggregation queries over encrypted relational databases. In DASFAA. Korea, 125-136.

[Halim09] Felix Halim. Panagiotis Karras. Roland H. C. Yap. Fast and effective histogram construction. In CIKM, pages 1167–1176, 2009.

[Herodotou11] Herodotos Herodotou, Shivnath Babu: Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. PVLDB 4(11): 1111-1122 (2011)

[Hore04] Bijit Hore, Sharad Mehrotra, Gene Tsudik. A Privacy-Preserving Index for Range Queries. VLDB, pp. 223-235, (2004)

[Hore12] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure multidimensional range queries over outsourced data. VLDB Journal. 21, 3, 333-358.

[Ioannidis03] Yannis Ioannidis. The history of histograms (abridged). In VLDB, pages 19–30, 2003.

[Jagadish98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, Torsten Suel. Optimal histograms with quality guarantees. In VLDB, pages 275–286, 1998.

[KISS12] INRIA, LIRIS, UVSQ, GEMALTO, CryptoExperts, CG78. 2012. Use cases and functional architecture specification, KISS deliverable ANR-11-INSE-0005-D1, 21/12/2012.

[Kissner05] Lea Kissner and Dawn Song. 2005. Privacy-Preserving Set Operations. In CRYPTO. 241–257.

[Lam07] H.Y. Lam, G.S.K. Fung, and W.K. Lee. 2007. A Novel Method to Construct Taxonomy Electrical Appliances Based on Load Signatures. IEEE Transactions on Consumer Electronics. 53, 2, 653-660.

[Liu10] Hongbo Liu, Hui Wang and Yingying Chen. 2010. Ensuring Data Storage Security against Frequency-based Attacks in Wireless Networks. In DCOSS. California, 201-215.

[Locher09] Thomas Locher. 2009. Foundations of Aggregation and Synchronization in Distributed Systems. ETH Zurich, isbn 978-3-86628-254-4.

[Mayberr12] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. PIRMAP: Efficient Private Information Retrieval for MapReduce. IACR Cryptology ePrint Archive, 2012:398, 2012.

[Molloy09] Ian Molloy, Ninghui Li, and Tiancheng Li. 2009. On the (in)security and (im)practicality of outsourcing precise association rule mining. In Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09). 872–877.

[Montjoye12] Yves-Alexandre de Montjoye, Samuel S Wang, Alex Pentland, Dinh Tien Tuan Anh, Anwitaman Datta. 2012. On the Trusted Use of Large-Scale Personal Data. IEEE Data Eng. Bull. 35, 4, 5-8.

[Muralikrishna88] M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In SIGMOD Conference, pages 28–36, 1988.

[Mykletun06] Einar Mykletun, and Gene Tsudik. 2006. Aggregation queries in the database-as-a-service model. In DBSec. France, 89-103.

[OJEC95] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data. Official Journal of the EC, 23, 1995.

[Paillier99] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In EUROCRYPT. 223-238.

[Poosala96] Viswanath Poosala, Yannis Ioannidis, Peter J. Haas, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In SIGMOD Conference, pages 294–305, 1996.

[Popa11] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In ACM SOSP. New York, 85-100.

[Rafaeli03] Sandro Rafaeli and David Hutchison. 2003. A Survey of Key Management for Secure Group Communication. ACM Computing Surveys. 35, 3, 309-329.

[Roy10] Indrajit Roy, Srinath T.V. Setty, Ann Kilzer, Vitaly Shmatikov, Emmett Witchel. Airavat: Security and privacy for MapReduce. USENIX NSDI, pp. 297–312, 2010.

[Ruan12] Anbang Ruan and Andrew Martin. TMR: Towards a trusted mapreduce infrastructure. IEEE World Congress on Services, pages 141–148, 2012.

[Song00] Dawn Xiaodong Song, David Wagner, Adrian Perrig. Practical techniques for searches on encrypted data. In Symposium on Security and Privacy, IEEE, pp. 44-55, 2000.

[StreamSQL15] StreamSQL. 2015. Available at : http://www.streambase.com/developers/docs/latest/streamsql/

[Tao03] Yufei Tao, Jimeng Sun, and Dimitris Papadias. 2003. Analysis of predictive spatiotemporal queries. ACM Transactions on Database Systems (TODS). 28, 4, 295–336.

[To13] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. 2013. Secure Global Protocol in Personal Data Server. SMIS Technical report. INRIA, France. http://www.cse.hcmut.edu.vn/~qcuong/INRIA/TechReport.pdf

[To14a] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. 2014a. Privacy-Preserving Query Execution using a Decentralized Architecture and Tamper Resistant Hardware. In EDBT. Athens, 487-498.

[To14b] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. 2014b. SQL/AA : Executing SQL on an Asymmetric Architecture. PVLDB. 7, 13, 1625-1628.

[To14c] Quoc-Cuong To, Benjamin Nguyen, Philippe Pucheral: Exécution sécurisée de requêtes avec agrégats sur des données distribuées. Ingénierie des Systèmes d'Information 19(4): 118-143 (2014)

[Tu13] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing analytical queries over encrypted data. PVLDB. 6, 5, 289–300.

[Wang06] Hui Wang and Laks V. Lakshmanan. 2006. Efficient secure query evaluation over encrypted xml database. In Proceedings of the 32nd International Conference on Very Large Data Bases.

[WEF12] The World Economic Forum. Rethinking Personal Data: Strengthening Trust. Industrial Report. May 2012.

[Wei09] Wei Wei, Juan Du, Ting Yu, Xiaohui Gu. SecureMR: A Service Integrity Assurance Framework for MapReduce. ACSAC, pp. 73–82, 2009.

[Wong07] Wai Kit Wong, David Cheung, Ben Kao and. Nikos Mamoulis. 2007. Security in outsourcing of association rule mining. In Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07). 111–122.

[Wu08] Bing Wu, Jie Wu, and Mihaela Cardei. 2008. A Survey of Key Management in Mobile Ad Hoc Networks. Handbook of Research on Wireless Security. 479-499.

[Wu11] Tsu-Yang Wu, Yuh-Min Tseng, and Ching-Wen Yu. 2011. Two-round contributory group key exchange protocol for wireless network environments. EURASIP Journal on Wireless Communications and Networking. 1, 1-8.

[Zhang11] Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen and XiaoFeng Wang. Sedic: privacy-aware data intensive computing on hybrid clouds. CCS 2011: 515-526.

[Zhang13] Xuyun Zhang, Chang Liu, Surya Nepal, Suraj Pandey, Jinjun Chen. A Privacy Leakage Upper-bound Constraint based Approach for Cost-effective Privacy Preserving of Intermediate Datasets in Cloud, IEEE Transactions on Parallel and Distributed Systems, 24(6): 1192-1202, 2013.

[Zhang14a] Chunwang Zhang, Ee-Chien Chang, Roland H.C. Yap. Tagged-MapReduce: A General Framework for Secure Computing with Mixed-Sensitivity Data on Hybrid Clouds. CCGrid, pp 31-40, 2014.

[Zhang14b] Xuyun Zhang, Laurence T. Yang, Chang Liu, Jinjun Chen. A Scalable Two-Phase Top-Down Specialization Approach for Data Anonymization Using MapReduce on Cloud. Parallel and Distributed Systems, IEEE Transactions on , vol.25, no.2, pp.363-373, 2014.