Secure Outsourced Data Analytics

Radu Ciucanu

Laboratoire d'Informatique Fondamentale d'Orléans (LIFO EA 4022)
Université d'Orléans – INSA Centre Val de Loire
ED Mathématiques, Informatique, Physique Théorique et Ingénierie des Systèmes (MIPTIS)

Secure Outsourced Data Analytics
Analyse Sécurisée de Données Externalisées

Habilitation à Diriger les Recherches (HDR) en Informatique

soutenue le 9 juin 2021 par

Radu Ciucanu

Composition du jury :

Présidente du jury :	Sara Bouchenak	Professeure, INSA Lyon / LIRIS
Rapporteurs :	Nicolas Anciaux	Directeur de Recherche, Inria Saclay
	Sébastien Gambs	Professeur, Université du Québec à Montréal, Canada
	Marie-Christine Rousset	Professeure, Université Grenoble Alpes / LIG
Examinateurs :	Claude Castelluccia	Directeur de Recherche, Inria Grenoble
	Pascal Lafourcade	MCF HDR, Université Clermont Auvergne / LIMOS
	Benjamin Nguyen	Professeur, INSA Centre Val de Loire / LIFO

Abstract. The goal of this document is to summarize my research activities since my recruitment as a Maître de Conférences in 2016. Most of my research in this period has been dedicated to addressing the data security concerns that occur when outsourcing some data to the cloud and then performing some computations (such as evaluating a query or a machine learning algorithm) on the data directly in the cloud. I focused on the honest-but-curious cloud model i.e., that executes tasks dutifully, but tries to gain as much information as possible. I contributed to: (i) secure evaluation of SPARQL queries on outsourced graphs, as well as synthetic generation of graphs and queries; (ii) secure protocols for sequential machine learning algorithms, with an emphasis on multi-armed bandits; (iii) secure MapReduce protocols for matrix multiplication and relational query evaluation. Our approach consists of developing distributed and secure protocols that combine existing algorithms with cryptographic schemes (such as AES and Paillier) and secure multi-party computations. Our protocols guarantee the same result as the standard, non-secure algorithms while enjoying desirable security properties.

Résumé. Le but de ce document est de résumer mes activités de recherche depuis mon recrutement en tant que Maître de Conférences en 2016. La plupart de mes recherches pendant cette période ont été consacrées aux problèmes de sécurité des données qui apparaissent lors de l'externalisation des données dans le cloud, suivie par certains calculs (tels que l'évaluation d'une requête ou d'un algorithme d'apprentissage automatique) directement dans le cloud. Je me suis concentré sur le modèle de cloud honnête-mais-curieux, c'est-à-dire qui exécute les tâches consciencieusement, mais essaie d'apprendre le plus d'informations possible. J'ai contribué à: (i) l'évaluation sécurisée de requêtes SPARQL sur des graphes externalisés, et sur la génération synthétique de graphes et requêtes; (ii) des protocoles sécurisés pour les algorithmes d'apprentissage automatique séquentiel, en mettant l'accent sur les bandits; (iii) des protocoles MapReduce sécurisés pour le produit matriciel et l'évaluation des requêtes relationnelles. Notre approche consiste à développer des protocoles distribués et sécurisés qui combinent des algorithmes existants avec des techniques cryptographiques (telles que AES et Paillier) et du calcul multipartite sécurisé. Nos protocoles garantissent le même résultat que les algorithmes standard non-sécurisés tout en bénéficiant de propriétés de sécurité.

Contents

Chapter 1

Introduction

Contents

The goal of this document is to summarize my research activities since my recruitment as a MCF (*Maître de Conférences*) in 2016. Most of my research in this period has been dedicated to addressing the *data security* concerns that occur when outsourcing some data to a public cloud and then performing some computations on the data directly in the cloud.

Indeed, outsourcing data and computations to a public cloud gained increasing popularity over the last years. Many cloud providers offer an important amount of data storage and computation power at a reasonable price e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure. However, cloud providers do not usually address the fundamental problem of data security. The outsourced data can be communicated over some network and processed on some machines where curious cloud admins could learn and leak sensitive data.

We depict the *database as a service* cloud computing service model in Figure 1.1, where a *data owner* outsources some database to the cloud, then a *data client* is allowed to submit some query to the cloud, which computes and returns the query's answers to the data client. Database as a service usually considers relational databases and data security is known as a major concern [CJP+11]: "*A significant barrier to deploying databases in the cloud is the perceived lack of privacy, which in turn reduces the degree of trust users are willing to place in the system.*" A typical solution to this concern (developed in systems such as CryptDB [PRZB11]) is to outsource encrypted data and use query-aware encryption schemes to answer queries directly on encrypted data. Moreover, if instead of a query, the cloud should evaluate some machine learning algorithm, we are in the context of *machine learning as a service* cloud computing model, for which the data security is also known as a major concern [BMMP18].

My recent research has been dedicated to proposing distributed and secure protocols that allow to do computations on outsourced data. I focused on the *honest-but-curious* cloud model i.e., that executes tasks dutifully, but tries to gain as much information as possible. This type of adversary is a reasonable assumption for public cloud providers. For instance, according to

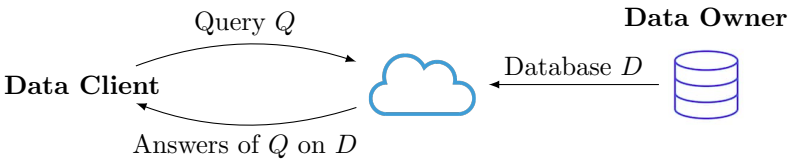


Figure 1.1: Outsourcing data and computations.

a recent survey on the security and privacy concerns of MapReduce [DDGS16], the honest-but-curious cloud is relevant for both security and privacy aspects:

- Security: one of the security threats is eavesdropping: "*an eavesdropping attack occurs when an adversary observes input data, intermediate outputs, the final outputs, and computations without any consent from the data and computation's owner.*"
- Privacy: one of the privacy challenges is data privacy protection from adversarial cloud providers: "*the goal of privacy in the presence of adversarial clouds is to minimize data leakage to the cloud provider while allowing users to perform operations on data.*"

The main technical part of this document is about my contributions on distributed and secure protocols, which are relevant to three classes of problems:

- *Graph query evaluation* [CL20b, CL20a]; my work on this topic benefited from my work on *synthetic graph and query generation* [BBC+17a, BBC+17b, BBC+16, ACM17a, ACM17b];
- *Sequential machine learning algorithms*, with an emphasis on multi-armed bandits [CLLS20, CDLS20, CLLS19];
- *MapReduce algorithms* for matrix multiplication [BCGL17, CGLY19c, CGLY19a] and relational query evaluation [CGLY19b, CGLY18, BCG+18].

To emphasize that my recent research on secure outsourced data analytics is a major thematic mobility with respect to my early research, I present an overview of my research activities before and after my MCF recruitment in Section 1.1 and 1.2, respectively.

1.1 Research before my MCF Recruitment (2012–16)

During my PhD and two subsequent international mobilities, I published my contributions in the *data management* community. Each of my three experiences was on a quite different topic, hence each geographic mobility was also a thematic mobility. Besides data management, I was also interested by *machine learning*, both during my PhD thesis (to learn queries from examples) and during my postdoc (to improve machine learning algorithms over compressed data).

- During my **PhD** [Ciu15] at Université Lille 1 / Inria Lille / CRIStAL UMR CNRS 9189 (2012–15), I worked on *query specification for non-expert users*. My approach was mainly based on learning queries from examples. My contributions span over three popular data models: relational, graphs, and XML. My main contributions are on *learning relational join queries* (research papers in TODS [BCS16] and EDBT [BCS14a], demo paper in VLDB [BCS14b]), *learning path queries on graph databases* (research [BCL15b] and demo [BCL15a] papers in EDBT), and *schema formalisms for unordered XML* (research papers in TOCS [BCS15] and DBPL [CS13]).

- During my **research visit** at the University of Toronto (2 months in 2015), I worked on an empirical evaluation of three *data integration* systems, which relied on the iBench metadata generator. This work is part of VLDB research [AGCM15] and demo [ACGM15] papers.
- During my **postdoc** at the University of Oxford (2015–16), I worked on *factorized databases* i.e., compressed representations of relational databases that exploit relational algebra properties and query structure. Our SIGMOD research paper [SOC16] relies on factorized databases to speedup the task of learning linear regression models over relational data.

1.2 Research after my MCF Recruitment (since 2016)

In Section 1.2.1, I briefly introduce my contributions that are at the core of this document, I explain the choice of each of the research topics on which I worked, and I outline the document organization. Then, I say a few words about my participation to student supervision and open source code in Section 1.2.2 and 1.2.3, respectively. I briefly discuss my teaching activities in Section 1.2.4, some of them being related to my research.

1.2.1 Overview of Research Topics and Contributions

In 2016, I became MCF at Université Clermont Auvergne / LIMOS UMR CNRS 6158. Since my contributions before my recruitment were relevant to the *data management* community, my first MCF contributions were also relevant to this community. More precisely, for working on **synthetic generation of graphs and queries**, I was motivated by the fact that during my PhD thesis, I was unable to find an off-the-shelf graph benchmark that I needed to empirically evaluate my theoretical contributions on learning path queries on graph databases [BCL15b]. Hence, I started to work on gMark, a synthetic generator of graphs and queries defined by UCRPQ (unions of conjunctions of regular path queries), which include recursive queries via the Kleene star. My main collaborators on gMark were my PhD advisers (Angela Bonifati and Aurélien Lemay), and an international collaborator (George Fletcher from TU Eindhoven). We finalized the gMark papers during my first MCF year: research paper in TKDE [BBC+17a], poster in ICDE [BBC+17b], and demo paper in VLDB [BBC+16]. Independently of my gMark collaborators, I worked on EGG, an extension of gMark that generates graphs with time-evolving properties, while co-advising the M2 internship of Karim Alami. Our ISWC demo paper [ACM17a] on EGG allowed to explore the *semantic Web* community.

Also during my first MCF year, I decided to do a major thematic mobility towards the *security* community, for three complementary reasons: (i) I was interested to get out of my comfort zone and explore new communities to have a wider vision on research in computer science. (ii) I met a LIMOS colleague, Pascal Lafourcade (expert in security and cryptography), who proposed me to co-advise the PhD thesis of Matthieu Giraud, who started in 2016. (iii) More pragmatically, my wife obtained a MCF position in 2017 at LIFO EA 4022, which motivated me to ask for a job mutation. Seen that the security is a major topic at LIFO (lab directed by Benjamin Nguyen), I considered that my thematic opening towards security would help me to get a mutation. When I started collaborating with Pascal Lafourcade, our first goal was to find a research topic at the crossroads of our complementary skills, and which makes sense in the context of cloud security i.e., the expected general topic of Matthieu Giraud's PhD. We decided to work on **secure MapReduce protocols** because MapReduce was a very popular big data paradigm and the data security is a real concern when the MapReduce algorithms are outsourced to the public cloud. Moreover, the choice of this topic allowed all of us to learn something new as none of us worked with MapReduce before. Our approach was to study standard MapReduce algorithms [LRU14,

Chapter 2], understand what are the needed computations, and look for practical cryptographic schemes that have desirable properties depending on the needed computations. Then, we put all pieces together to propose secure protocols that should give exactly the same output as the standard MapReduce algorithms, while hiding the input and output from the honest-but-curious cloud. We worked on problems such as matrix multiplication and relational query evaluation (joins, grouping and aggregation, set intersection), which led to papers in ARES [BCGL17], SECRYPT/ICETE [CGLY18, CGLY19c, CGLY19b, CGLY19a], and FPS [BCG+18].

In 2018, I obtained a mutation at INSA Centre Val de Loire / LIFO, which did not preclude my productive collaboration with Pascal. In addition to finalizing our works on secure MapReduce, we aimed at securing other types of problems. Seen that machine learning is one of the trendiest topics in computer science, we naturally thought at tackling the data security issues that occur when outsourcing machine learning algorithms to the cloud. Machine learning is a very broad topic, hence in order to identify interesting problems to which we can contribute, we started collaborating with Marta Soare from LIFO. Marta is expert on sequential learning i.e., a learning setting that is widely used for decision making under uncertainty. In collaboration with her and with Pascal, we contributed to **secure protocols for multi-armed bandits**, a class of sequential learning algorithms where a learning agent needs to sequentially decide which "arm" to choose among several options (with unknown reward) available in the environment. The multi-arm bandits have practical applications such as Web advertisement, where the arms are the advertised items and the rewards are given by the user click-through rate. To secure multi-armed bandit algorithms, an important difference with respect to securing MapReduce is that we no longer have a master controller in charge of distributing the tasks among the nodes, hence we need to define our own strategy for distributing the input data and computation tasks. In our first paper (ISPEC [CLLS19]), we proposed a protocol for best arm identification, which guarantees that (i) none of the cloud nodes can learn enough information to infer the best arm, and moreover (ii) our algorithm yields exactly the same output as the standard non-secure algorithm. In our next papers (TrustCom [CLLS20] and ProvSec [CDLS20]), we proposed protocols with guarantees similar to (i) and (ii), but for the task of cumulative reward maximization. We considered both bandits with simple structure (Bernoulli reward distributions) and more complex structure (linear bandits). In particular, our secure protocol for linear bandits [CDLS20] was developed by Anatole Delabrouille during his M2 internship that we co-advised. We are still actively working on the topic of secure protocols for multi-armed bandits.

In addition to the aforementioned research topic, I am currently contributing to **secure evaluation of graph queries**. To explain how I started this topic, I should mention that at fall 2019, when I was MCF for already three years, I decided to take a retrospective look at my research activities spanning this period. I noticed that I worked on quite diverse topics, from synthetic graph and query generation, to secure and distributed protocols for both MapReduce and bandit problems. Hence, I naturally wanted to find synergies between the different topics. This is how I started to work on the GOOSE distributed framework for secure graph outsourcing and SPARQL evaluation, and I proposed to Pascal Lafourcade to collaborate with me. GOOSE enjoys desirable security properties that are similar to the ones from the previous paragraphs e.g., the cloud cannot learn more than limited pieces of the input and output. The class of SPARQL queries supported in GOOSE is precisely the class of UCRPQ supported in gMark. Moreover, I relied on gMark to realize a large-scale empirical evaluation of GOOSE in order to understand its strong and weak points. We published GOOSE as a research paper in DBSec [CL20b], a conference at the crossroads between the data management and security communities. Moreover, since GOOSE is an innovative system that allows secure data outsourcing and query evaluation relevant to popular semantic Web technologies (RDF and SPARQL), we showcased GOOSE to the semantic Web community, via our ISWC demo [CL20a]. I plan to extend GOOSE in the near

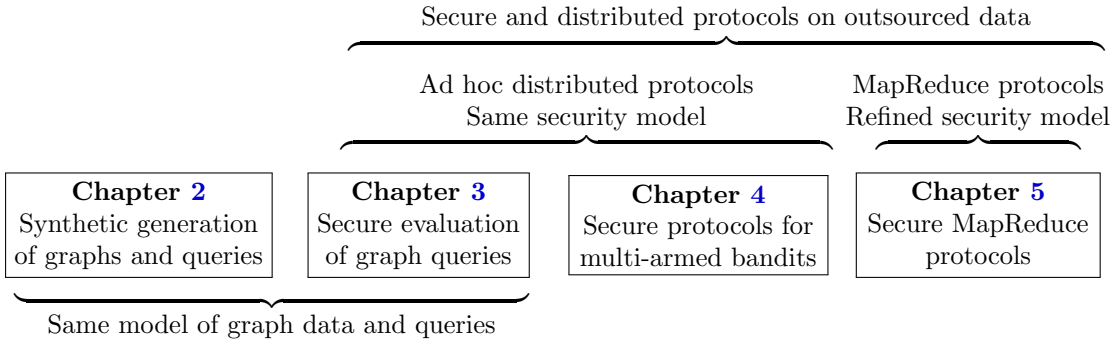


Figure 1.2: Links between the four core chapters.

future to support classes of queries richer that UCRPQ, but I need to first extend gMark in order to rely on finely tuned graphs and queries for the future empirical evaluations.

Document organization. The core of this document consists of four chapters synthesizing my contributions on four research topics: synthetic generation of graphs and queries (Chapter 2), secure evaluation of graph queries (Chapter 3), secure protocols for multi-armed bandits (Chapter 4), and secure MapReduce protocols (Chapter 5). These chapters have a similar structure:

- Context
- Related work
- Summary of contributions
- Conclusions

In each of these chapters, I report only on results that are already published. Hence, instead of focusing on the details of each result, my goal was to provide an overview of the context behind the work and on the main scientific results. For instance, although the majority of my papers have empirical evaluations, I do not exhaustively include all of them in this document, but I rather selected two representative ones, which correspond to the open source codes for which I am the main developer (cf. Section 1.2.3).

All my research was done through collaborations, therefore in these four chapters I use "we" rather than "I" when describing my research, except for the "Conclusions" sections of each chapter, when I briefly outline my contribution to each of the collaborative works.

I present my future research perspectives in Chapter 6. Finally, Chapter 7 contains the document's references, out of which Section 7.1 contains my list of international publications.

Links between the four core chapters. The ordering of the four core chapters does not match the chronological order of the underlying research, which is rather 2, 5, 4, 3. I briefly justify the ordering choice, by relying on the common points and differences between the contributions, summarized in Figure 1.2. I start with the synthetic generation of graphs and queries (Chapter 2) because this is the only connection between my research before and after becoming MCF. Then, I chose to continue with the secure evaluation of graph queries (Chapter 3) right after the synthetic generation of graphs and queries because both chapters rely on the same model of graph data and queries. Moreover, I chose to include the secure protocols for multi-armed bandits (Chapter 4) right after the secure evaluation of graph queries because both chapters have in common the approach of proposing ad hoc distributed protocols, and rely on the same classical security model cf. [Gol04, Chapter 7] (where *honest-but-curious* is denoted *semi-honest*), in particular

(i) each cloud node is trusted: it correctly does the required computations, it does not sniff the network and it does not collude with other nodes, and (ii) an external observer has access to all messages exchanged over the network. Relying on ad hoc distributed protocols in Chapters 3 and 4 makes easier to enforce the no-collusion hypothesis from the classical honest-but-curious model. However, for our work on secure MapReduce protocols (Chapter 5), the distribution strategy is not ad hoc, but is constrained by the MapReduce paradigm. This means that a same cloud node that is used to store a chunk of the input data may be also used as a computing node for map and/or reduce functions at some further round, which may imply a collusion. Consequently, we believe that it is important to consider collusions in our cloud model for secure MapReduce, which means that the security model from Chapter 5 is refined with respect to the security model from Chapters 3 and 4. Despite relying on slightly different security models as briefly justified above and developed through the document, the three Chapters 3, 4, and 5 have in common the approach of proposing secure and distributed protocols on outsourced data, the main topic of this document.

1.2.2 Student Supervision

I realized some of the works presented in this document while supervising the following students:

- *Anatole Delabrouille*: Université de Bordeaux, M2 internship (6 months in 2019–20), co-advised with Pascal Lafourcade (Université Clermont Auvergne / LIMOS) and Marta Soare (Université d'Orléans / LIFO);

  Topic: Secure protocols for multi-armed bandits;

  Publication together: [CDLS20].

- *Matthieu Giraud*: Université Clermont Auvergne, PhD at LIMOS (2016–19), co-advised with Pascal Lafourcade;

  Topic: Secure MapReduce protocols;

  Publications together: [CGLY19a, CGLY19c, CGLY19b, CGLY18, BCG+18, BCGL17].

- *Karim Alami*: Université Clermont Auvergne, M2 internship (6 months in 2016–17), co-advised with Engelbert Mephu Nguifo (Université Clermont Auvergne / LIMOS);

  Topic: Evolving graph generation;

  Publications together: [ACM17a, ACM17b].

1.2.3 Open Source Code

In addition to contributing to the design and theoretical analysis of my papers, I also participated to prototype and experimentally evaluate the different contributions. Hence, some of my papers are accompanied by *open source code*, whose main goal is to allow the researchers from the community to reproduce our experimental results. Next, I give pointers to public Git repositories relevant to research described in this document, as well as my participation.

- Main developer

  [CL20b, CL20a] `https://github.com/radu1/goose`

  [CLLS20] `https://github.com/radu1/secure-ucb`

- Adviser of students who developed the code during their internships

[CDLS20] https://github.com/anatole33/LinUCB-secure

[ACM17a, ACM17b] https://github.com/karimalami7/EGG

- Other participations

  [CLLS19] https://gitlab-sds.insa-cvl.fr/vciucanu/secure-bai-in-mab-public-code

  [BBC+17a, BBC+17b, BBC+16] https://github.com/gbagan/gmark

1.2.4 Beyond Research

Since 2016, I am teaching on average 230h/year, which includes the compulsory service of 192h and a few extra hours. My teaching activities were dedicated to a diverse audience, from Bac+1 to Bac+5, in engineering school and BSc/MSc, to computer science students and students from other departments, in French and in English. Although I had to majorly revise my teaching service twice (in 2017 due to the major revision of teaching programs in the newly created Université Clermont Auvergne, and in 2018 due to my mutation at INSA Centre Val de Loire), I can roughly classify my teaching activities in the following categories:

- *Standard introductory computer science courses* e.g., algorithms, programming, databases.
- *Advanced courses related to data management* e.g., Big data, NoSQL, XML, Data mining, Business intelligence. The development of such advanced courses was often related to my research activities. For example, my research on gMark inspired me to teach graph databases and (recursive) graph queries. Moreover, I started to teach the MapReduce paradigm at the same time that I started to co-advise a PhD thesis on secure MapReduce protocols. In 2020, I started to teach the Spark paradigm (which becomes increasingly more popular than MapReduce), and I started to teach practical works on secure execution of sequential learning algorithms, in a strong relationship with my research.
- *Other teaching-related activities* e.g., supervising students from my teaching department when they are pursuing internships in industry. Moreover, in 2017–18, I was responsible of foreign relations of the engineering school ISIMA Clermont, an interesting but time-consuming administrative duty.

Chapter 2

Synthetic Generation of Graphs and Queries

Massive graph data sets are pervasive in contemporary application domains. Hence, graph database systems are becoming increasingly important. In the experimental study of these systems, it is vital that the research community has shared solutions for the generation of database instances and query workloads having predictable and controllable properties. We propose gMark (graph benchMark), a domain- and query language-independent generator of graphs and query workloads. A core contribution of gMark is its ability to control the diversity of properties of the generated graph instances and query workloads coupled to these instances. gMark is the first generator that supports (i) unions of conjunctions of regular path queries, a fundamental graph query paradigm including recursive queries, and (ii) schema-driven query selectivity estimation, a key feature in controlling workload chokepoints. Since the nodes and edges of large-scale graphs have properties that naturally evolve over time, we proposed EGG (Evolving Graph Generator), which relies on gMark as a building block. EGG generates evolving graphs based on finely tuned temporal constraints. The goal of this chapter is to present an overview of the design and implementation principles of both gMark and EGG systems.

Relevant Publications

- Research paper: TKDE journal [BBC+17a] (on gMark)
- Demonstrations of prototypes: VLDB [BBC+16] (on gMark), ISWC [ACM17a] (on EGG)
- Other: ICDE poster [BBC+17b] (on gMark), TDLSG@ECML/PKDD [ACM17b] (on EGG)

Contents

2.1 Context

We study the problem of schema-driven generation of synthetic graph instances and corresponding query workloads for use in experimental analysis of graph database systems. Our study is motivated by the ubiquity of graph data in modern application domains, such as social and biological networks, and geographic databases, to name a few. In response to these pressures, systems that can handle massive graph-structured data sets are under active research and development. These systems span from pure graph database systems to more focused knowledge representation systems. Native graph databases such as Neo4j [Neo] propose their own declarative data model and query language, with particular attention to query optimization and performance. In contrast to this trend of specialized systems, general-purpose systems such as LogicBlox [AtG+15] rely on declarative solutions that can cover a broader range of use cases. Furthermore, knowledge representation systems such as Virtuoso [Vir] and Apache Jena [Jen] implement the standard RDF graph data model and SPARQL query language to handle complex navigational and recursive queries on large-scale semantic Web data.

Synthetic graph and query generation, and benchmarking solutions for graph data management systems have been a proliferating activity, which started within the semantic Web community [AHÖD14, SHLP09] and recently within the database community [EALP+15]. The latter has been the first to propose a chokepoint-driven design of graph database benchmarks. By relying on a fixed schema and a carefully designed set of benchmark queries, this lets the community focus on crucial features of query optimization and/or parallel processing, and decreases the confusion and the incomparable results of evaluated systems. We propose a complementary approach in which the focus is not on the design of individual queries but rather on whole query workloads. Our approach relies on the control of diversity of both graph schemas and query workloads, which lets us vary the structural properties of data as well as tailor the generated queries to a particular domain or application. We emphasize that a workload-centric approach primarily targets different benchmarking and experimental scenarios from the query-centric approach of current benchmarks. Indeed, we believe that our approach is important in contexts in which multiple queries (i.e., those belonging to a query workload) need to be considered altogether, which occurs e.g., in multi-query optimization and workload-driven database tuning.

gMark is a synthetic generator that realizes the aforementioned workload-centric perspective. gMark takes a schema-driven approach to the flexible and tightly-controlled generation of synthetic graphs coupled with sophisticated query workloads. The most notable novel features with respect to the existing solutions are the support for recursive queries and the query selectivity estimation in the generated query workloads. EGG is an extension of gMark that generates a sequence of graph snapshots satisfying the graph evolution constraints specified by the user.

Motivating example. Assume that a user wants to perform an extensive empirical evaluation of a new graph query processing algorithm that she designed. For this purpose, the user needs to efficiently generate: (i) graphs of different characteristics and sizes (to test the robustness and scalability of her algorithm), and (ii) query workloads sufficiently diverse to highlight strong or weak points of her new development. Additionally, our user would like to specify all parameters in a declarative way and to be able to simulate real-world scenarios.

For instance, the user would like to generate graphs simulating a bibliographical database that uses a simple schema consisting of 5 node types and 4 edge predicates. Intuitively, the database consists of *researchers* who *author papers* that are *published* in *conferences* (held in *cities*) and that can be *extended* to *journals*. Moreover, the user would like to specify constraints on the number of occurrences for both the node types and edge predicates, either as proportions of the total size of the graph or as fixed numbers e.g., as in Figure 2.1(a) and 2.1(b). For instance,

Node type	*Constraint*
`researcher`	50%
`paper`	30%
`journal`	10%
`conference`	10%
`city`	100 (fixed)

(a) Node types.

Edge predicate	*Constraint*
`authors`	50%
`publishedIn`	30%
`heldIn`	10%
`extendedTo`	10%

(b) Edge predicates.

source type $\xrightarrow{\textit{predicate}}$ *target type*	*In-distribution*	*Out-distribution*
`researcher` $\xrightarrow{\texttt{authors}}$ `paper`	Gaussian	Zipfian
`paper` $\xrightarrow{\texttt{publishedIn}}$ `conference`	Gaussian	Uniform [1,1]
`paper` $\xrightarrow{\texttt{extendedTo}}$ `journal`	Gaussian	Uniform [0,1]
`conference` $\xrightarrow{\texttt{heldIn}}$ `city`	Zipfian	Uniform [1,1]

(c) In- and out-degree distributions.

Figure 2.1: The bibliographical motivating example.

for graphs of arbitrary size, half of the nodes should be authors, but a fixed number of nodes should be cities where conferences are held. Indeed, in a realistic scenario the number of authors increases over time, whereas the number of cities remains more or less constant.

Moreover, our user wants to specify real-world relationships between types and predicates via schema constraints e.g., as in Figure 2.1(c). For instance, the first line encodes that the number of authors on papers follows a Gaussian distribution (the *in-distribution* of the schema constraint), whereas the number of papers authored by a researcher follows a Zipfian (power-law) distribution (the *out-distribution* of the schema constraint). The following lines in Figure 2.1(c) encode constraints such as: a paper is published in exactly one conference, a paper can be extended or not to a journal, a conference is held in exactly one city, the number of conferences per city follows a Zipfian distribution, etc. As we discuss in Section 2.2, the state-of-the-art graph generators do not allow to specify such finely tuned graph constraints, and moreover, no not support the query features that we consider in gMark.

2.2 Related Work

Data generation and benchmarking frameworks have played an important role in database systems research over the last decades, where efforts such as the TPC benchmarks and XML benchmarking suites have been crucial for advancing the state of the art. Patterson [Pat12] stated that "*when a field has good benchmarks, we settle debates and the field makes rapid progress*".

Similarly, in support of the experimental study of graph data management solutions, a variety of synthetic graph tools such as SP2Bench [SHLP09], LDBC [EALP+15], LUBM [GPH05], BSBM [BS09], and WatDiv [AHÖD14] have been developed in the research community. We next discuss the positioning of gMark with respect to these existing solutions.

A first difference is that the state-of-the-art generators either rely on fixed graph schemas (SP2Bench, LDBC, LUBM, BSBM) or have limited schema support (WatDiv). For example, in SP2Bench (that is also based on a bibliographical scenario as our motivating example), all constraints are hardcoded and the only parameter that a user can specify is the size of the graph, which makes it impossible for the user to finely tune schema-related characteristics of the graph. In WatDiv, although the user can specify global constraints on the node types and the

out-distributions, the absence of global constraints on the edge predicates and the absence of in-distributions limit the selectivity control of the queries from the generated query workloads. On the other hand, constraints such as those outlined in the motivating example of Section 2.1 can be easily declaratively specified as an input gMark *graph configuration* (cf. Figure 2.2) via a few lines of XML. To the best of our knowledge, there is no other graph generator where such constraints can be specified.

Additionally, the state-of-the-art generators provide limited or no support for generating tailored query workloads to accompany graph instances. In particular, SP2Bench, LDBC, LUBM, and BSBM rely on fixed sets of queries. Moreover, WatDiv supports a workload-centric approach, by doing the selectivity estimation on the generated graph instances, which becomes unfeasible when dealing with massive graphs and query workloads. Hence, it does not provide the fine-grained level of control of query behavior as gMark, where we address the challenge of selectivity estimation by generating tailored workloads directly from the graph schema, where query selectivity is set as one of the input parameters. In general, we are not aware of any solutions for controlling selectivity during query generation relying solely on graph schemas. In gMark, we allow the user to finely tune the selectivities of the generated queries. For instance, the user can specify that she wants queries that, for any graph size, have constant, linear, or quadratic selectivity. We define these selectivity classes in [BBC+17a] and give some intuitions in Section 2.3.1. To the best of our knowledge, no synthetic generator supports such a feature.

As another remarkable difference to the state-of-the-art generators, none of them supports recursive queries such as $(\texttt{authors}\cdot\texttt{authors}^{-})^{*}$ which selects all pairs of researchers linked by a co-authorship path (by $^{-}$ we denote the predicate inverse and by * the transitive closure). As shown in the input gMark *query workload configuration* in Figure 2.2, the user can finely tune e.g., the structure and selectivity of such queries. To the best of our knowledge, gMark is the first solution for generating workloads exhibiting recursive path queries. In particular, the queries generated by gMark are the so-called *Unions of Conjunctions of Regular Path Queries* (UCRPQ) [BFVY18]. This fundamental query language covers graph queries that appear in practice e.g., in SPARQL 1.1 and openCypher. UCRPQ are also expressible in modern Datalog-like query languages [AtG+15] and in SQL. gMark supports the output of query workloads in all these concrete query language syntaxes.

The limitations discussed thus far for (static) graph generators translate for evolving graph generators. Indeed, there exist very few evolving graph generators (e.g., EvoGen [MP16] that extends LUBM [GPH05]), but to the best of our knowledge, there does not exist any schema-driven evolving graph generator proposed before our EGG system.

2.3 Summary of Contributions

We present an overview of gMark in Section 2.3.1 and of EGG in Section 2.3.2.

2.3.1 gMark

gMark is an open source[1] schema-driven generator of graphs and queries. In this section, we briefly discuss our gMark contributions, which are all developed in [BBC+17a]. We first present the problems of schema-driven graph and query workload generation, which are both intractable (NP-complete) in general. Hence, to efficiently generate graphs and queries, we chose to develop a gMark generation algorithm that has a linear running time (in the size of the input+output) and sometimes relaxes some of the constraints. As for the generated queries, the most notable

[1] https://github.com/gbagan/gmark

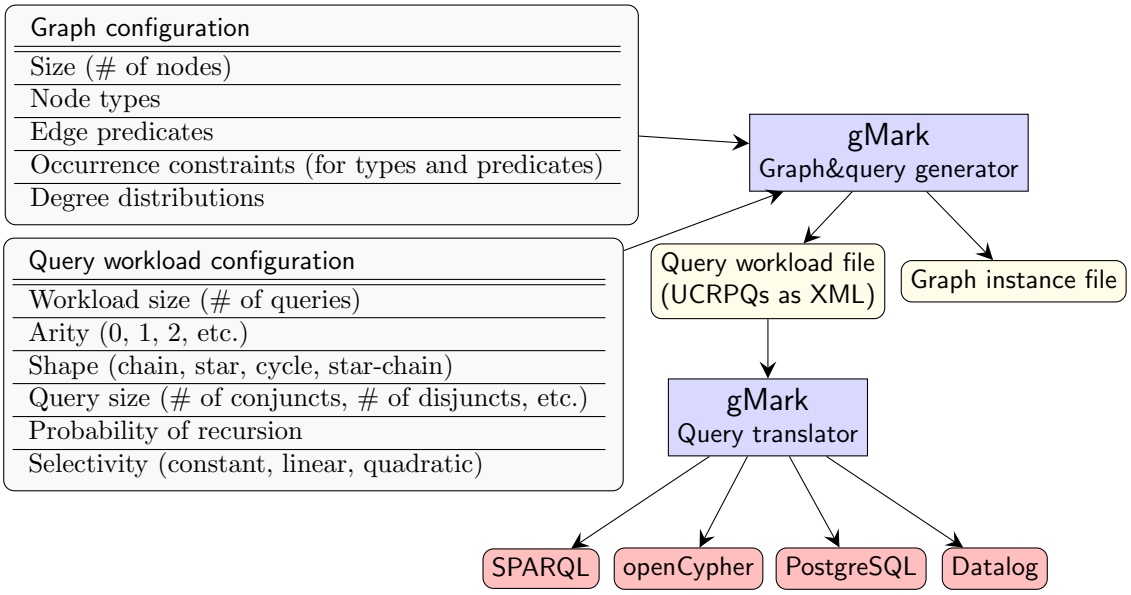


Figure 2.2: Overview of the gMark workflow.

novel features are support for recursive queries and schema-driven query selectivity estimation. Moreover, we discuss the capability of gMark to cover diverse graphs and query workloads, the accuracy of the estimated selectivities, and the scalability of the generator. We relied on gMark to perform an experimental comparison of state-of-the-art graph query engines, which brings to light limitations of such engines, in particular with respect to recursive query processing.

Graph generation. An important gMark design principle was to make it *domain-independent*, such that it can be used to target a rich variety of realistic domains. Hence, to finely tune the characteristics of the generated graphs, gMark has a built-in support for schema definition, which allows users to specify fundamental schema constraints.

More precisely, a graph database instance generated by gMark is a *directed, edge-labeled graph* $G = (V, E)$, where V is a set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of directed edges between nodes of V and with labels (aka predicates, we use these two terms interchangeably) from an *alphabet* Σ. The gMark graph generation algorithm takes as input a *graph configuration* (cf. Figure 2.2), which includes the desired graph size, as well as schema constraints. We have already intuitively illustrated all these constraints via our motivating example from Section 2.1, as recalled next:

- *Node types* i.e., an enumeration of the allowed node types, knowing that each node of the generated graph is associated with exactly one type. On our example, this corresponds to the first column in Figure 2.1(a).
- *Edge predicates* i.e., an enumeration of the allowed edge predicates in Σ. On our example, this corresponds to the first column in Figure 2.1(b).
- *Occurrence constraints* i.e., each node type/edge predicate is associated with either a proportion of its occurrences or a fixed constant value. On our example, this corresponds to the second column in Figure 2.1(a) and the second column in Figure 2.1(b).
- *Degree distributions* i.e., a triple (source type, predicate, target type) is associated with a pair of in- and out-degree distributions. On our example, this corresponds to Figure 2.1(c). A degree distribution is a probability distribution, among which gMark supports uniform,

Gaussian (aka normal) and Zipfian distributions. For each distribution, the user may specify the relevant *parameters* i.e., min and max for uniform, μ and σ for Gaussian, and s for Zipfian. If the user wants to specify only the in- or out-distribution, she can mark the other one as non-specified.

Notice that all aforementioned constraints that are allowed in a graph configuration need to be consistent in order to guarantee the compatibility of the number of generated ingoing and outgoing edges. Unfortunately, as we prove in [BBC+17a]: *It is NP-complete to decide whether there exists a graph satisfying a given graph configuration.*

Consequently, it is not always possible to generate in polynomial time a graph satisfying a given graph configuration. Nevertheless, efficiently generating graph instances is an important goal in designing a graph generator, essentially because of *scalability*. Hence, due to practical reasons, we decided that gMark should always return a graph to the user instead of performing a potentially costly satisfiability check and possibly aborting the generation. To this purpose, we must therefore relax some constraints specified in the graph configuration. Hence, the gMark *graph generation algorithm* (detailed in [BBC+17a]) takes a heuristic approach that guarantees a linear running time in the size of the input+output. Due to the heuristic nature of our algorithm, some of the graph configuration constraints might not be fulfilled in the output graph: it may happen that the generated graph does not satisfy the precise values of the in- or out-distributions (such distributions are essential in the NP-completeness proof). However, our algorithm preserves the types of distributions (uniform, Gaussian, Zipfian) as specified in the graph configuration, even though the generated number of edges may not satisfy the exact parameters of the distributions. Our choice of preserving the global distributions in the graph generator turns out to be useful in the query generator, where our selectivity estimation technique relies on the types of distribution (uniform, Gaussian, Zipfian) and not on their exact parameters.

Query generation. Regarding the query generation, the two important gMark design principles were (i) *language- and system-independence*, and (ii) *controlled query workload diversity*.

To achieve (i), gMark generates queries in an abstract syntax (UCRPQ = *Unions of Conjunctions of Regular Path Queries*), serialized in XML format. Then, the gMark query translator (which is independent of the query generator) translates the generated queries into concrete query languages (SPARQL, SQL, Datalog, openCypher).

To define UCRPQ, recall that Σ is an alphabet and let $\Sigma^+ = \{a, a^- \mid a \in \Sigma\}$, where a^- denotes the *inverse* of the edge label a. By $?x, ?y, \ldots$ we denote variables. A *query rule* is an expression of the form *head* $\leftarrow$ *body*, more precisely:

$$(\overline{?v}) \leftarrow (?x_1, r_1, ?y_1), \ldots, (?x_n, r_n, ?y_n)$$

where: for each $1 \leq i \leq n$, it is the case that $?x_i, ?y_i$ are variables and $\overline{?v}$ is a vector of zero or more of these variables, the length of which is called the *arity* of the rule. For each $1 \leq i \leq n$, it is the case that r_i is a regular expression over Σ^+ using $\{\cdot, +, *\}$ (i.e., *concatenation*, *disjunction*, and *Kleene star*). Without loss of generality, we restrict regular expressions to only use recursion (i.e., the Kleene star symbol $*$) at the outermost level. Hence, regular expressions can always be written to take either the form $(P_1 + \cdots + P_k)$ or the form $(P_1 + \cdots + P_k)^*$, for some $k > 0$, where each P_i is a *path expression* i.e., a concatenation of zero or more symbols in Σ^+. We say that each $(?x_i, r_i, ?y_i)$ of the body is a *conjunct*. A *query* $Q \in$ UCRPQ is a finite non-empty set of query rules of the same arity, each rule having several *conjuncts*, each conjunct having several *disjuncts* whose paths have a certain *length*. For example take the following UCRPQ query:

$$(?x, ?y, ?z) \leftarrow (?x, (a \cdot b + c)^*, ?y), (?y, a, ?w), (?w, b^-, ?z)$$
$$(?x, ?y, ?z) \leftarrow (?x, (a \cdot b + c)^*, ?y), (?y, a, ?z)$$

This query selects nodes $?x, ?y, ?z$ such that one can navigate between $?x$ and $?y$ with a path in the language of $(a \cdot b + c)^*$, and moreover, can navigate between $?y$ and $?z$ with a path in the language of $a \cdot b^- + a$. This query consists of two rules consisting of 3 conjuncts and 2 conjuncts, respectively The conjuncts of the form $(?x, (a \cdot b + c)^*, ?y)$ have two disjuncts (of length 2 and 1, respectively) and all other conjuncts have only one disjunct (of length 1).

To achieve the design principle (ii) i.e., controlled query workload diversity, gMark supports a broad range of parameters in the so-called *query workload configuration*. We stress that gMark query generation additionally takes as input the graph configuration, in order to make the generated queries meaningful to the generated graph. Hence, a natural consequence of the intractability of the graph generation is that: *It is NP-complete to decide whether there exists a query workload satisfying a given pair of graph and query workload configurations.*

Regarding the parameters from the query workload configuration (cf. Figure 2.2), they are capable of estimating both navigational and recursive query execution performance, while not relying on a fixed set of query templates. To the best of our knowledge, existing solutions for graph databases do not support such configurable range of parameters. The first three parameters have self-explanatory names and we need to say a few words about the others. We define the *query size* as a tuple $([c_{\min}, c_{\max}], [d_{\min}, d_{\max}], [l_{\min}, l_{\max}])$ providing intervals of minimal and maximal values for the number conjuncts, disjuncts, and length of the paths in the query, respectively. The aforementioned example query has size $([2, 3], [1, 2], [1, 2])$. In gMark, the user can specify minimal and maximal values for all these parameters; in turn, the query generation algorithm assigns values that range in these intervals. Moreover, the *probability of recursion* is the probability to have a Kleene star above a disjunct.

We also need to say a few words about the *selectivity* constraints. Assume a collection of graph configurations where only the graph size changes, in other words the node types, edge predicates, occurrence constraints, and degree distributions are fixed. Given a binary query Q, for every graph G satisfying one of the aforementioned graph configurations, we assume that the number of answers returned by evaluating Q on G behaves asymptotically as a function of the form $\beta|G|^\alpha$, where α and β are real constants. We say that the above value α is the selectivity value of Q. The selectivity value of a query is by definition bounded by the query arity. We focus here on binary queries, hence we have selectivity values such that $0 \leq \alpha \leq 2$. We identify three practical query classes, depending on whether α is closer to 0, 1, or 2:

- *Constant queries* (for which $\alpha \approx 0$) select a number of results that does not grow (or barely grows) with the graph size. For instance, a query selecting pairs (`country`, `language`) is constant if the graphs follow a realistic schema specifying that the numbers of countries and languages do not grow with the graph size, and hence the number of query results is more or less constant.
- *Linear queries* (for which $\alpha \approx 1$) select a number of results that grows at a rate close to the growth of the number of nodes in the graph instances. For example, a query selecting pairs (`language`, `user`) is linear if the schema specifies that the number of users grows with the graph, whereas the number of languages is more or less constant. Another example of a linear query selects pairs (`user`, `address`) if we assume that the schema specifies that each user has precisely one address and the number of users grows linearly with the graph.
- *Quadratic queries* (for which $\alpha \approx 2$) select a number of results that grows at a rate close to the growth of the square of the number of nodes in the graph instances. For example, the transitive closure of the `knows` predicate in a social network is quadratic because a realistic schema should specify that this predicate follows a power-law Zipfian in- and out-distribution. Thus, the query result contains Cartesian products of subsets of users that know and are known by some hub users of the social network.

We propose in [BBC+17a] a *query workload generation algorithm*, which required the development of a selectivity algebra that generalizes the aforementioned intuitions. Based on a graph configuration, the idea is to infer which are the possible selectivities that can be obtained by navigating between two node types, and then to compose such selectivities in order to satisfy the input query size, shape, and recursion constraints. Our schema-driven selectivity estimation is to the best of our knowledge the first of its kind. In the current version of gMark, the selectivity estimation is supported only for binary queries. This already implied the development of a non-trivial algorithm, and, as future work, we plan to extend it for queries with arbitrary arity.

Empirical evaluation. To illustrate the ability to encode user-defined constraints across a variety of application domains, we encoded 4 use cases (*bib, shop, social-network, uniprot*), which are available on the GitHub repository of gMark. More precisely, *bib* is a bibliographical use case, in the spirit of the motivating example from Section 2.1. Moreover, to show that it is easy to adapt scenarios of existing state-of-the-art benchmarks into meaningful gMark configurations, we point out that the *shop* use case is the gMark encoding of Waterloo SPARQL Diversity Test Suite (WatDiv) [AHÖD14], whereas the *social-network* use case is the gMark encoding of the LDBC Social Network Benchmark [EALP+15]. Each use case encodes different types of constraints, hence the generated graphs and queries have different characteristics. These use cases were particularly helpful for the empirical evaluation of GOOSE [CL20b], as detailed in Section 3.3.3.

Furthermore, in [BBC+17a], we provide a detailed empirical evaluation of gMark, using a diversity of use cases, with respect to (i) *accuracy*: we generated constant/linear/quadratic queries and we compared the selectivities observed in practice vs the constraints specified in the query configuration, to conclude similar asymptotic behaviors, and (ii) *scalability*: we generated graphs of increasing size and we observed a linear running time, as expected from our theoretical analysis. For instance, on a laptop, for graphs of 100M nodes, the graph generation time is in the order of minutes. As for query generation, gMark takes in the order of seconds to generate a workload of a thousand queries and translate them in all four supported practical syntaxes.

Moreover, in [BBC+17a], we relied on gMark to perform an experimental comparison of state-of-the-art graph query engines. We compared 4 systems, supporting SPARQL, SQL, Datalog, and openCypher, respectively. Note that not all systems support the full expressive power of UCRPQ. In particular, arbitrary UCRPQ can be expressed in SPARQL, SQL, and Datalog, while openCypher supports only those UCRPQ having no occurrences of inverse or concatenation under Kleene star. Some of the generated recursive queries do indeed exhibit inverse and/or concatenation in a recursive conjunct. In these cases, the corresponding openCypher query has only the non-inverse symbol and/or the first symbol in a concatenation of symbols, respectively. Furthermore, while all other languages adopt the classical homomorphic semantics for conjunctive queries, openCypher adopts an isomorphic semantics. For these two reasons, openCypher queries often have answer sets that differ from their counterparts in the other languages.

Our experimental comparison of state-of-the-art graph query engines led to some interesting results: (i) for *non-recursive queries* (i.e., no Kleene stars in the generated queries), the SQL engine is typically the most efficient, and (ii) for *recursive queries* (i.e., the generated queries contain Kleene stars), we observed that the Datalog engine was the most efficient. We believe that our empirical evaluation brought to light limitations of current graph engines, seen that the generic SQL and Datalog systems typically gave the best results compared to the dedicated graph systems. Moreover, gMark was very useful in this study because it provided us diverse graphs and query workloads.

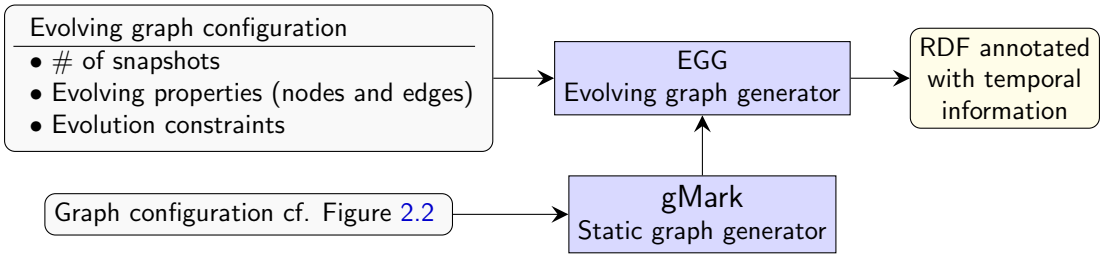


Figure 2.3: Overview of the EGG workflow.

2.3.2 EGG

EGG is an open source[2] system that generates evolving graphs from finely tuned temporal constraints, while relying on gMark as a building block. Similarly to gMark, EGG is schema-driven and domain-independent. We depict the architecture of EGG in Figure 3.3. EGG takes as input (i) an initial graph generated by gMark, and (ii) an evolving graph configuration that encodes how the evolving properties of the node types and edge predicates from a gMark configuration should evolve over time. The output of EGG is an RDF graph annotated with temporal information (in the spirit of [TB09]) that encodes a sequence of graph snapshots satisfying the constraints given by the user. We use next as running example a geographical database, but we have been additionally able to easily encode different domains such as a social network, a DBLP-like bibliographical network, or an online shop.

Static graph configurations. Assume that a user wants to generate graphs simulating a geographical database storing data about cities, and different facilities such as transportation and hotels. The user can specify as gMark input the following types of constraints: (i) graph size, given as # of nodes; (ii) node types e.g., `city` and `hotel`, and edge types e.g., `train` and `contains`; (iii) occurrence constraints e.g., 10% of the graph nodes should be of type `city`, whereas 90% of the graph nodes should be of type `hotel`; (iv) degree distributions e.g.,

source type $\xrightarrow{\textit{predicate}}$ *target type*	*In-distribution*	*Out-distribution*
`city` $\xrightarrow{\texttt{contains}}$ `hotel`	Uniform [1,1]	Zipfian

meaning that we can have an edge of type `contains` from a node of type `city` to a node of type `hotel`, with a Zipfian out-distribution since it is realistic to assume that the number of hotels in a city follows such a power-law distribution, and moreover, a uniform (which in this particular case is a constant 1) in-distribution since a hotel is located in precisely one city.

We call such gMark graph configurations as being static since the nodes of type e.g., `city` and `hotel` are rarely created or deleted. Nonetheless, such nodes (as well as the different edges connecting them) possess properties that naturally evolve over time, in an interdependent manner. The user can specify such evolving properties as input of EGG, as we illustrate next.

Evolving graph configurations. Assume that our user generates with gMark a graph having nodes of type `city` and `hotel`, and edges of type `train` (connecting two cities) and `contains` (connecting a city to a hotel). Next, the user wants to add properties that evolve over time for the aforementioned nodes and edges, assuming that a graph snapshot corresponds to a day. We

[2] https://github.com/karimalami7/EGG

next give examples of such properties, together with finely tuned constraints to evolve among consecutive snapshots. A node of type `hotel` has the following evolving properties:

- `availableRooms` (quantitative discrete), which can have as values integers in the interval [1,100], following a binomial distribution. There is a probability of 80% that it changes from a snapshot to the next one, and it can increment or decrement by an integer up to 5 between two consecutive snapshots.
- `star` (ordered qualitative), which can have five possible values, following a geometric distribution. It can only change every thirty snapshots, with a probability of 10%, and it can only increment or decrement by 1.
- `hotelPrice` (quantitative continuous), whose values follow a normal distribution in an interval that is dynamically constructed based on the value of the property `star`. Moreover, `hotelPrice` is anti-correlated with `availableRooms` i.e., if `availableRooms` decreases, then `hotelPrice` increases, and vice-versa.

The user can similarly specify evolving properties and evolution constraints for the node type `city` (e.g., properties `weather` and `airQuality`) and for the edge type `train` (e.g., property `trainPrice`). It is worth noting that we allow the EGG user to specify *validity properties* i.e., Boolean properties encoding whether a given node or edge exists at a given snapshot e.g., a train connection between two cities may not be valid during all snapshots.

Implementation challenges. Building a system like EGG is an ambitious goal since we allow the user to specify very expressive constraints. We next discuss two interesting challenges.

(i) *Computational complexity.* As shown earlier, we allow the specification of evolution constraints where the value of a property among consecutive snapshots depends on another property. We model the inter-dependencies between such evolving properties with a dependency graph. If the aforementioned dependency graph is cyclic, the generation algorithm may not halt. Hence, in our implementation we require that the dependency graph is acyclic and we sort it topologically to decide in which order we the evolution constraints.

(ii) *Storage redundancy.* A naive solution to store the generated evolving graphs would be to entirely store each snapshot, which would yield a redundant storage due to the graph parts that are static throughout the snapshots. To minimize such redundancy, we rely on a storage format inspired by [TB09] that uses named graphs to express temporal information in RDF. Our output format (that we serialize using the TriG syntax[3]) allows us to decouple the storage of the static parts of the graph (i.e., structural information satisfied in all snapshots) and the evolving parts of the graph (i.e., the property values that change from a snapshot to the next one). For example, we use named graphs of the form

```
ns1:G30 {<hotel:27> ns2:hasProperty <Property:hotelPrice>.}
ns1:G31 {<hotel:27> ns2:hasProperty <Property:availableRooms>.}
```

encoding that the hotel 27 has properties `hotelPrice` and `availableRooms`. Moreover, for each graph snapshot, we have a further named graph where each of the named graphs of the form above has associated a value e.g.,

```
ns1:snapshot1 {ns1:G30 ns2:value "45.5".  ns1:G31 ns3:value "57".}
ns1:snapshot2 {ns1:G30 ns2:value "47.9".  ns1:G31 ns3:value "54".}
```

Empirical evaluation. For showing the accuracy of EGG and its sensitivity to different constraints, we implemented a web interface to illustrate that the generated graphs match the

[3] https://www.w3.org/TR/trig/

input constraints. To emphasize the ease of realizing empirical evaluations on top of EGG, we performed a performance comparison of two approaches for answering historical reachability queries [SPL15], which ask whether there exists a path between two nodes in a specified interval of time. We provide in a wiki on our GitHub page of EGG more details on all experiments that we performed: accuracy and scalability of EGG, as well as a comparison of approaches for answering historical reachability queries.

2.4 Conclusions

We presented an overview of our work on synthetic generation of graphs and queries. gMark was a collaboration with colleagues from LIRIS (Guillaume Bagan, Angela Bonifati), Inria Lille (Aurélien Lemay), and TU Eindhoven (George Fletcher, Nicky Advokaat). Our TKDE journal paper [BBC+17a] presents the design, implementation, and empirical study of gMark, and is accompanied by an extended abstract presented as an ICDE poster [BBC+17b] and a VLDB demo paper [BBC+16]. My contributions on gMark are both theoretical (I formalized the problems of graph and query generation, and proved that both of them are NP-complete) and practical (I proposed the XML specification of the gMark use cases; moreover, for the sake of the demo, I implemented a Web interface of gMark, which allows users to visualize the generated queries and their translations to practical query languages). Furthermore, EGG was developed by Karim Alami, during his M2 internship, that I co-advised at LIMOS with Engelbert Mephu Nguifo. My contributions are mainly conceptual as I advised Karim during the entire research workflow from problem formalization to empirical evaluation. EGG led to an ISWC demo paper [ACM17a] and a workshop paper [ACM17b].

An important point is that gMark already had some impact on the community seen that it already received more than 90 citations according to Google Scholar[4]. gMark was particularly useful for empirical studies for some of my subsequent research on secure query evaluation of outsourced graphs [CL20b] (as detailed in Chapter 3), and also on empirical studies of systems developed in different research groups e.g., [JGGL20].

Looking ahead to the future work, there are many directions for further investigation, which are strongly connected to the scenarios where gMark could be useful to improve the existing empirical evaluations. For example, while working on [CL20b], we observed similar points of improvement as the authors of [JGGL20] when they performed their empirical evaluation. More precisely, we need to extend gMark to be able to generate classes of queries beyond UCRPQ, some of the desirable query features being to support UCRPQ where some of the query nodes are constants, or to combine UCRPQ with aggregates. Extending gMark to support such query features is not a trivial problem, seen that graph and query generation are already NP-complete for the currently supported types of constraints. It is very important to be able to continue tuning the query selectivities for the new query features in order to be able to generate queries with controllable properties.

[4] https://scholar.google.fr/citations?user=1WXDq-sAAAAJ

Chapter 3

Secure Evaluation of Graph Queries

We address the security concerns that occur when outsourcing graph data and query evaluation to an honest-but-curious cloud. We propose GOOSE, a secure framework for Graph OutsOurcing and SPARQL Evaluation, which relies on cryptographic schemes and secure multi-party computation to achieve desirable security properties: (i) no cloud node can learn the graph, (ii) no cloud node can learn at the same time the query and the query answers, and (iii) an external network observer cannot learn the graph, the query, or the query answers. As query language, GOOSE supports unions of conjunctions of regular path queries that are at the core of the W3C's SPARQL 1.1, including recursive queries. In this chapter, we present the workflow of GOOSE via a running example, as well as an overview of the theoretical analysis and empirical evaluation of GOOSE.

Relevant Publications

- Research paper: DBSec [CL20b]
- Demonstration of prototype: ISWC [CL20a]

Contents

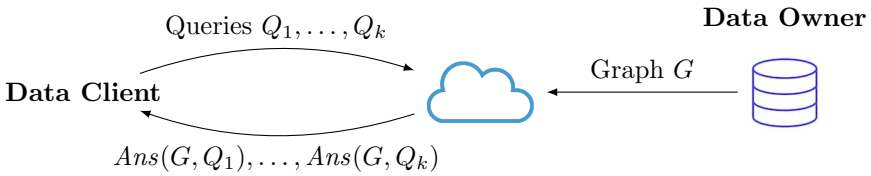


Figure 3.1: Outsourcing graph data and query evaluation.

3.1 Context

We address the data security issues that occur when outsourcing an RDF graph database to a public cloud and querying the outsourced graph with SPARQL. We depict the considered scenario in Figure 3.1, where a *data owner* outsources a graph to the cloud, then a *data client* is allowed to query the graph by submitting queries to the cloud, which computes and returns the queries' answers to the data client. Our scenario is inspired by the *database as a service* cloud computing service model, which usually considers relational databases, and security is well-known as a major concern [CJP+11], as already outlined in Section 1. A typical solution to this concern (developed in systems such as CryptDB [PRZB11]) is to outsource encrypted data and use query-aware encryption schemes to answer queries directly on encrypted data.

Although SQL and SPARQL share some common functionalities, adapting a system such as CryptDB to securely answer SPARQL queries on outsourced graphs is not trivial because SPARQL allows to naturally express classes of queries that are cumbersome to express in SQL. This is the case for the recursive queries, which can be easily expressed using the Kleene star in the property paths of SPARQL 1.1[1]. To express such recursive queries in SQL, one needs to define recursive views. After analyzing the source code of the SQL parser inside CryptDB[2], we concluded that such queries are beyond the scope of CryptDB and it is unclear how hard it is to extend their system to support recursive queries.

We propose GOOSE, a framework for Graph OutsOurcing and SPARQL Evaluation, which allows the data owner to securely outsource to the cloud a graph that can be then queried by the data client. We assume that the cloud is *honest-but-curious* i.e., executes tasks dutifully, but tries to gain as much information as possible about the data, queries, and query answers. Similarly to CryptDB, GOOSE evaluates queries on encrypted data without any change to the query engine, which in our case is the standard Apache Jena for evaluating SPARQL queries. As query language, GOOSE supports *Unions of Conjunctions of Regular Path Queries* (UCRPQ) that are at the core of the W3C's SPARQL 1.1, including recursive queries.

The key ingredients of GOOSE are: (i) *secure multi-party computation* i.e., the graph storage is distributed among three cloud participants, which can jointly compute the query answers for each submitted query, but none of the cloud participants can learn the graph, and none of the cloud participants can learn at the same time the query and the answers of the query on the graph, and (ii) *cryptographic schemes* i.e., all messages exchanged between GOOSE participants are encrypted with AES-CBC [AES01, BDJR97] such that an external network observer cannot learn the graph, the query, or the answers of the query on the graph. The challenge of building GOOSE was to efficiently distribute storage and computation among as few cloud participants as possible, while minimizing the time needed for cryptographic primitives.

[1] https://www.w3.org/TR/sparql11-property-paths/
[2] https://css.csail.mit.edu/cryptdb/#Software

3.2 Related Work

Our work on GOOSE follows a recent line of research on tackling the security and privacy concerns related to RDF graph data storage and querying (see [KVd18] for a survey).

The main novelty of GOOSE with respect to the state-of-the art is that we consider UCRPQ, whereas the related works [FKPS18, FKPS17, KSAK13, KAMD13, Gie05] focus on non-recursive SPARQL queries defined as triple patterns. In particular, this is the case for HDT$_{crypt}$ [FKPS18] that is to the best of our knowledge the state-of-the-art system for query evaluation on encrypted graphs. HDT$_{crypt}$ combines HDT (a compression technique useful for reducing RDF storage space) and encryption (to hide particular subgraphs from unauthorized users). Our work is complementary to this related research direction since we assume that query evaluation is outsourced to the cloud and our security goals are different from theirs: we want to avoid that the cloud nodes and network observer learn the entire graph, queries or query answers, whereas their goal is to allow multiple users with different access rights to query the graph. A common idea between HDT compression and our GOOSE is to map nodes and edge labels to integers, but for different goals. For HDT$_{crypt}$, the goal is to reduce storage and bandwidth usage. For GOOSE, the combination of this technique with secure multi-party computation is particularly useful for achieving security since the actual mapping functions are not shared with the node responsible for query evaluation, which is able to evaluate UCRPQ without knowing which are the true nodes and edge labels that it manipulates.

If one chooses to store RDF graphs in a relational database and query them with SQL, then one can choose CryptDB [PRZB11] to run queries directly in the encrypted domain. As already mentioned, CryptDB does not currently support recursive queries, and such queries are anywise cumbersome to express in SQL as they require recursive views. CryptDB has been extended as CryptGraphDB [ALC18] to run Neo4j queries on encrypted graphs, while supporting the same query features as in CryptDB, hence without considering recursive queries. GOOSE is complementary to these systems since our goal is to propose a system that is able to run UCRPQ while enjoying similar security properties.

We chose to rely on UCRPQ because this class of queries is at the core of the W3C's SPARQL 1.1 property paths, including recursive queries via the Kleene star. A recent large-scale analytical study of SPARQL query logs [BMT20] includes more than a million such recursive queries, which suggests that a secure protocol for evaluating recursive graph queries would be also useful in practice. To the best of our knowledge, GOOSE is the first provably-secure system that is able to run UCRPQ on outsourced graphs, without doing any change to the standard SPARQL engine.

Our work is also related to query-based linked data anonymization [DBRT18], where the idea is that the data owner, before publishing a graph, adds some noise, specified declaratively using SPARQL. Then, a data client is able to download the anonymized graph and query it. However, their hypothesis and ours are different as we assume that the bulk of computations is outsourced to the cloud and our data client does not need to do any computation effort other than decrypting the query answers received from the cloud. For us, the challenge is to design a distributed protocol that guarantees that the cloud cannot learn the graph, queries, and query answers, while minimizing the overhead due to cryptographic primitives. On the other hand, their challenge is to anonymize the graph before publishing, while finding a good compromise between privacy and utility.

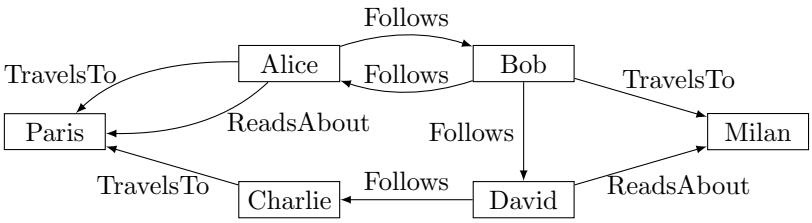


Figure 3.2: Example of graph database.

3.3 Summary of Contributions

We present the workflow of GOOSE via a running example in Section 3.3.1. We briefly discuss the theoretical analysis of GOOSE in Section 3.3.2. We report on the empirical evaluation of GOOSE in Section 3.3.3.

3.3.1 Workflow of GOOSE

Before outlining GOOSE via a running example, we need to first define graph data and queries.

Graph data. An RDF (Resource Description Framework[3]) graph database is a set of triples (s, p, o) where s is the *subject*, p is the *predicate*, and o is the *object*. According to the specification, $s \in \mathcal{I} \cup \mathcal{B}, p \in \mathcal{I}$, $o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$, where $\mathcal{I}, \mathcal{B}, \mathcal{L}$ are three disjoint sets of Internationalized Resource Identifiers (IRIs), blank nodes, and literals, respectively. For our purposes, the distinction between IRIs, blank nodes, and literals is not important. Therefore, we simply assume that a graph database $G = (V, E)$ is a *directed, edge-labeled graph*, where V is a set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of directed edges between nodes of V and with labels from an *alphabet* Σ. For example, the graph in Figure 3.2 has:

- Set of nodes $V = \{$Alice, Bob, Charlie, David, Milan, Paris$\}$. The first four nodes correspond to persons and the last two correspond to cities.
- Alphabet $\Sigma = \{$Follows, ReadsAbout, TravelsTo$\}$. The first label occurs between two persons and defines the follower relation as in a social network e.g., Twitter. The other two labels occur between a person and a city.
- Set of edges E such as (Alice, Follows, Bob), (Alice, TravelsTo, Paris), etc. There are 9 edges in total, corresponding to the 9 arrows in Figure 3.2.

Graph queries. We focus on *Unions of Conjunctions of Regular Path Queries* (UCRPQ), which are at the core of the W3C's SPARQL 1.1[4]. We have already defined UCRPQ in Section 2.3.1 when we introduced the gMark query generation. For the sake of this section, we additionally denote by $Ans(G, Q)$ the answers of query Q over a graph G, using standard SPARQL semantics. For example, the UCRPQ

$$(?x, ?z) \leftarrow (?x, \text{Follows}^+, ?y), (?y, \text{TravelsTo}, ?z)$$

selects nodes $?x, ?z$ such that there exists node $?y$ such that one can go from $?x$ to $?y$ with a path in the language of "Follows$^+$" and can go from $?y$ to $?z$ with a path in the language of "TravelsTo". The answers of this query on the graph from Figure 3.2 are (Alice, Milan), (Alice,

[3] https://www.w3.org/TR/rdf11-concepts/
[4] https://www.w3.org/TR/sparql11-query/

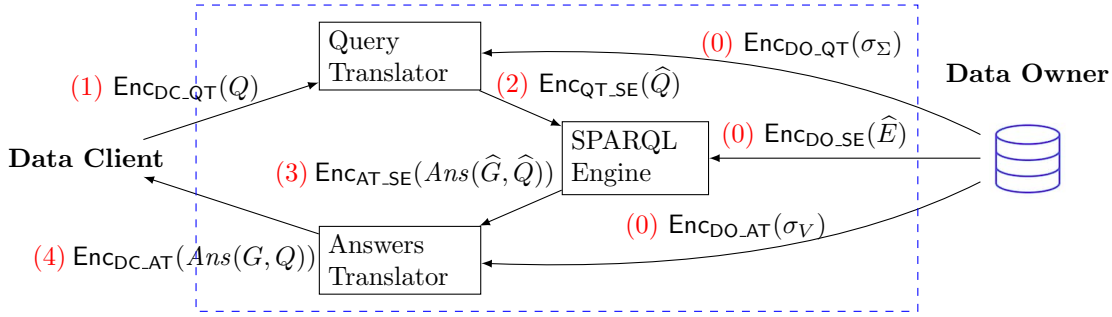


Figure 3.3: Architecture of GOOSE. The dashed rectangle is the cloud.

Paris), (Bob, Milan), (Bob, Paris), (David, Paris). For example, the tuple (Alice, Paris) is an answer because of paths Alice $\underrightarrow{\text{Follows}}$ Bob $\underrightarrow{\text{Follows}}$ David $\underrightarrow{\text{Follows}}$ Charlie and Charlie $\underrightarrow{\text{TravelsTo}}$ Paris, where $?x, ?y, ?z$ are mapped to Alice, Charlie, Paris, respectively.

Architecture of GOOSE. In Figure 3.3, we depict the architecture of GOOSE, which has 5 participants: *data owner* (DO), who owns the graph that it outsources to the cloud in order to be queried, *data client* (DC), who submits graph queries to the cloud and receives query answers, and 3 cloud participants: *query translator* (QT), *SPARQL engine* (SE) and *answers translator* (AT). By Enc_{A_B} or Enc_{B_A} we denote symmetric AES-CBC encryption using the key [AES01] shared between participants A and B. We have 7 such shared keys because there are 7 combinations of participants exchanging messages, hence 7 arrows in Figure 3.3. We assume that the sharing of AES keys has been done before starting the actual protocol and there are many classical key exchange protocols in the literature for doing this.

Step 0. The *graph outsourcing* (i.e., the 3 outgoing arrows from DO in Figure 3.3) is done only once at the beginning by DO. Intuitively, DO sends to each cloud participant a piece of the graph such that each participant can perform its task during query evaluation but no participant can reconstruct the entire graph. As shown in the pseudocode of DO in Figure 3.4(a), DO generates two random bijections: σ_Σ and σ_V, one for the edge labels and another one for the graph nodes, respectively. By σ^{-1} we denote the inverse of σ (this is needed later on at the end of query evaluation). For our example graph in Figure 3.2, DO may generate:

$$\sigma_V = \{\text{Alice} \to 5, \text{Bob} \to 3, \text{Charlie} \to 0, \text{David} \to 1, \text{Milan} \to 2, \text{Paris} \to 4\}$$
$$\sigma_\Sigma = \{\text{Follows} \to 1, \text{ReadsAbout} \to 2, \text{TravelsTo} \to 0\}.$$

Then, DO uses these two functions to hide the graph edges. As shown in Figure 3.4(a), by $\widehat{E}$ we denote the hidden set of edges generated from E, where the nodes are replaced using σ_V, and the edge labels are replaced using σ_Σ. On our example graph in Figure 3.2, edge (Alice, Follows, Bob) becomes (5, 1, 3), edge (Alice, ReadsAbout, Paris) becomes (5, 2, 4), etc., and finally:

$$\widehat{E} = \{(5,1,3), (5,2,4), (5,0,4), (3,1,5), (3,1,1), (3,0,2), (0,0,4), (1,1,0), (1,2,2)\}.$$

As shown in Figure 3.3 and 3.4(a), DO sends σ_Σ, σ_V, and $\widehat{E}$ to cloud nodes QT, AT, and SE, respectively. Each message sent over the network is encrypted with the key shared between DO and the corresponding cloud participant, which can decrypt the message upon reception. Messages are encrypted to avoid that an external observer that sees them in clear is able to learn

let σ_Σ = random bijection : $\Sigma \to \{0, \ldots, |\Sigma|-1\}$
let σ_V = random bijection : $V \to \{0, \ldots, |V|-1\}$
let $\widehat{E} = \{(\sigma_V(s), \sigma_\Sigma(p), \sigma_V(o)) \mid (s, p, o) \in E\}$
send $\mathsf{Enc}_{\mathsf{DO\_QT}}(\sigma_\Sigma)$ to QT
send $\mathsf{Enc}_{\mathsf{DO\_AT}}(\sigma_V)$ to AT
send $\mathsf{Enc}_{\mathsf{DO\_SE}}(\widehat{E})$ to SE

(a) Pseudocode of outsourcing graph $G = (V, E)$ by DO (step 0).

generate query $\widehat{Q}$ from Q by replacing each occurrence of a label p with $\sigma_\Sigma(p)$
send $\mathsf{Enc}_{\mathsf{QT\_SE}}(\widehat{Q})$ to SE

(b) Pseudocode of QT during query evaluation (step 2).

let $\widehat{G} = (\bigcup_{(\widehat{s},\widehat{p},\widehat{o})\in\widehat{E}}\{\widehat{s}, \widehat{o}\}, \widehat{E})$
let $Ans(\widehat{G}, \widehat{Q})$ be the answers of $\widehat{Q}$ on $\widehat{G}$, computed with some SPARQL engine
send $\mathsf{Enc}_{\mathsf{AT\_SE}}(Ans(\widehat{G}, \widehat{Q}))$ to AT

(c) Pseudocode of SE during query evaluation (step 3).

let $Ans(G, Q) = \{(\sigma_V^{-1}(v_1), \ldots, \sigma_V^{-1}(v_n)) \mid (v_1, \ldots, v_n) \in Ans(\widehat{G}, \widehat{Q})\}$
send $\mathsf{Enc}_{\mathsf{DC\_AT}}(Ans(G, Q))$ to DC

(d) Pseudocode of AT during query evaluation (step 4).

Figure 3.4: Pseudocode of the non-trivial steps of GOOSE cf. Figure 3.3.

the graph G, thus violating one of the desirable security properties (that we state in Section 3.3.2). Moreover, the distribution of graph storage among cloud participants makes that none of them can learn the graph G, which is also a desirable security property cf. Section 3.3.2.

We next discuss *query evaluation* i.e., steps 1–4 cf. Figure 3.3, done for each query submitted by DC. Similarly to graph outsourcing, each message exchanged over the network during query evaluation is encrypted with the key shared between corresponding participants, such that an external observer cannot learn the query and its answers to satisfy another desirable security property cf. Section 3.3.2.

Step 1. DC submits query Q to QT. For example, recall the aforementioned query

$$(?x, ?z) \leftarrow (?x, \text{Follows}^+, ?y), (?y, \text{TravelsTo}, ?z).$$

Step 2. QT translates the received query Q by replacing all labels used in Q using the function σ_Σ received from DO, as shown in Figure 3.4(b). By $\widehat{Q}$ we denote the query Q translated using σ_Σ. On our running example, the query from step 1 becomes $(?x, ?z) \leftarrow (?x, 1^+, ?y), (?y, 0, ?z)$.

Step 3. As shown in Figure 3.4(c), SE evaluates translated query $\widehat{Q}$ received from QT at step 2 on the graph with hidden nodes and edge labels as defined by $\widehat{E}$ received from DO during step 0. To do so, SE simply uses some standard SPARQL engine as a black-box, without any change to the query engine. We denote the result of SE by $Ans(\widehat{G}, \widehat{Q})$, where the true answers $Ans(G, Q)$ are still hidden using function σ_V. On our running example, $Ans(\widehat{G}, \widehat{Q}) = \{(5, 2), (5, 4), (3, 2), (3, 4), (1, 4)\}$.

Step 4. AT uses the function σ_V^{-1} to translate the received hidden query answers $Ans(\widehat{G}, \widehat{Q})$ into the true query answers, as shown in Figure 3.4(d). On our running example, AT recovers $Ans(G, Q)$ ={(Alice, Milan), (Alice, Paris), (Bob, Milan), (Bob, Paris), (David, Paris)} that AT sends to DC.

3.3.2 Theoretical Analysis of GOOSE

Security. We assume that the cloud is honest-but-curious. More precisely, as already mentioned at the end of Section 1.2.1, we follow the classical formulation in [Gol04, Chapter 7] (where *honest-but-curious* is denoted *semi-honest*), in particular (i) each cloud node is trusted: it correctly does the required computations, it does not sniff the network and it does not collude with other nodes, and (ii) an external observer has access to all messages exchanged over the network.

By $data_A$, we denote the data to which A has access, where A can be a cloud participant (QT, SE, AT) or an external network observer (*ext*). In [CL20b], we characterize *data* for all participants and we formally prove that GOOSE satisfies the following *security properties*:

- For each cloud participant $A \in \{\mathsf{QT}, \mathsf{SE}, \mathsf{AT}\}$, A cannot guess from $data_A$ the graph $G = (V, E)$ with probability better than random under the assumption that bijections σ_Σ and σ_V are pseudorandom.
- For each cloud participant $A \in \{\mathsf{QT}, \mathsf{SE}, \mathsf{AT}\}$, A cannot guess from $data_A$, at the same time, a query Q and its answers $Ans(G, Q)$ with probability better than random under the assumption that bijections σ_Σ and σ_V are pseudorandom.
- Given $data_{ext}$, then the graph $G = (V, E)$, any query Q, and any query answers $Ans(G, Q)$ are *indistinguishable* of random for an external network observer of GOOSE under the assumption that the symmetric encryption used is IND-CPA.

Intuitively, we achieve these properties by exchanging only encrypted messages, and moreover, by distributing the computations among several cloud node participants, each of them having access only to the specific data that it needs for performing its task and nothing else. The proofs of the first two properties rely on the assumption that bijections σ_Σ and σ_V used for graph outsourcing are pseudorandom. The proof of the third property relies on the IND-CPA property of AES-CBC [AES01, BDJR97].

Complexity. In [CL20b], we analyze the theoretical complexity of GOOSE, by quantifying the number of calls of cryptographic primitives. We show that GOOSE uses a number of AES-CBC encryptions/decryptions that is linear in the input's and output's size.

Correctness. In [CL20b], we point out a reduction from GOOSE to the standard SPARQL evaluation engine used as a black-box in SE. The reduction relies on the consistency property of AES-CBC.

3.3.3 Empirical Evaluation of GOOSE

We report on a large-scale empirical evaluation (also presented in [CL20b]), which is devoted to showing the feasibility and scalability of GOOSE, for both graph outsourcing and query evaluation. We also compare GOOSE query evaluation with standard SPARQL evaluation and we zoom on the running time shares of each GOOSE participant.

Implementation. We implemented GOOSE in Python 3. For AES-CBC we used keys of 256 bits with the *PyCryptodome* library[5]. As SPARQL engine, we used Apache Jena [Jen]. We carried out our experiments on a system with CPU Intel Xeon of 3GHz and 755GB of RAM, running CentOS Linux 7. We stress that the large RAM is simply a characteristic of the server that we had at our disposal for our experiments, hence the large RAM is not a constraint imposed by GOOSE. Except for the largest scaling factor (10^7 cf. Figure 3.5), we were also able to run entire workflows of GOOSE on a laptop with 16GB of RAM.

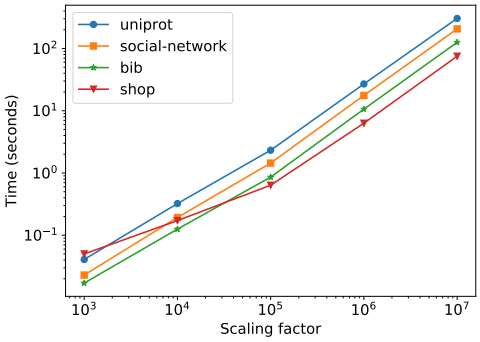
Open source code. For reproducibility reasons, we make available on a public GitHub repository[6] our source code, together with scripts that install needed libraries, run the large-scale experiment, and generate the plots. This experiment took 8 days on our system and generated 46GB of data (total size for graphs, queries, and query answers).

Datasets. We relied on gMark (see Chapter 2 for an overview), which provided us a large quantity of diverse data and queries to stress-test GOOSE. We used all 4 use cases from the gMark repository: *uniprot* (biological data where proteins interact with other proteins, are encoded on genes, etc.), *shop* (online shop selling different types of products to users, etc.), *social-network* (social network where persons know other persons, work in companies, etc.), and *bib* (bibliographical data about researchers that author papers published in journals or conferences, etc.). Each use case encodes different types of constraints, which make the generated graphs and queries have different characteristics, that we detail when necessary to explain experimental results.

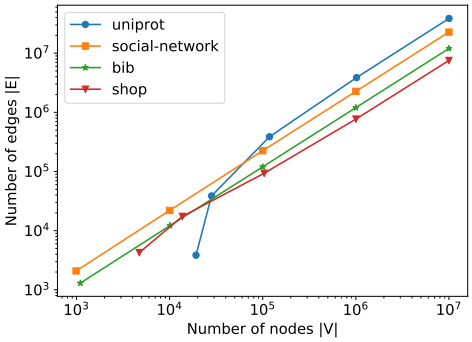
We performed three experiments: (i) scalability of graph outsourcing, (ii) scalability of query evaluation, and (iii) zoom of end-to-end solution, which consists of outsourcing a graph and answering several queries on it. Our experiments confirmed the linear time behavior expected by our theoretical analysis, and showed the scalability and feasibility of GOOSE. An interesting observation is that the bottleneck of GOOSE is the standard SPARQL engine used as a black-box, and not the use of cryptographic operations. The limitation of current SPARQL engines for evaluating recursive queries, has been already pointed out in [BBC+17a].

Scalability of graph outsourcing. For each of the 4 use cases, we consider 5 scaling factors, from 10^3 to 10^7, where a scaling factor n means that gMark should generate a graph with n nodes. For each combination (use case, scaling factor), we report the GOOSE graph outsourcing time, averaged over 10 graphs, each of them outsourced 3 times. We show the result of this experiment in Figure 3.5(a), where we observe a smooth, linear-time behavior. We next explain the running times difference between the use cases by detailing their characteristics. In particular, the number of generated nodes for a scaling factor depends on how large is n and what constraints are specified in the use case. This is why, to help understanding the behavior in Figure 3.5(a), we also plot in Figure 3.5(b) the size (# of nodes vs # of edges) for the generated graphs. To simulate realistic graph constraints, each use case specifies how the number of nodes of some type increases: there are types of nodes whose number increases when the graph size increases (e.g., users and purchases in *shop*), and types of nodes whose number is constant for all graph sizes (e.g., cities and countries in *shop*). When we take a small scaling factor and a use case with strong constraints on the types with constant number of occurrences, gMark may have to add nodes beyond the size specified by the scaling factor to satisfy the number of nodes for each type. This explains the behavior for small scaling factors in Figure 3.5(b) for *shop* (12 types of constant node types), and *uniprot* (3 types of constant node types, among which one with 15K

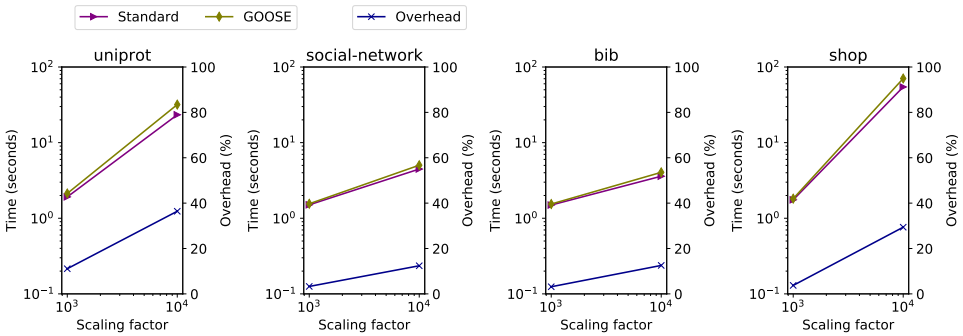
[5] https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html
[6] https://github.com/radu1/goose



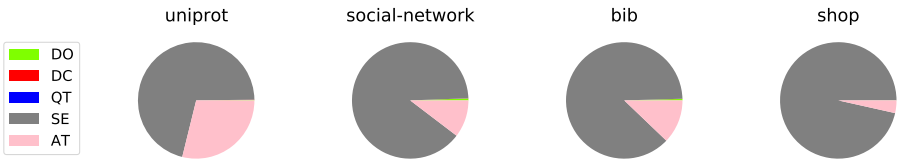
(a) Scalability of graph outsourcing.



(b) Size of graphs in dataset.



(c) Scalability of query evaluation, and comparison standard vs GOOSE.



(d) Zoom on end-to-end solution i.e., graph outsourcing and query evaluation for a workload of 5 queries, for graphs of fixed scaling factor 10^4. The shares of participants DO, DC, and QT are barely visible.

Figure 3.5: Experimental results on GOOSE.

occurrences, hence for the scaling factor 10^3, the generated graphs have at least 15 times more nodes). For large scaling factors $(10^5, 10^6, 10^7)$, the number of constant node types is dominated by the nodes with types that increase with the graph size, hence the hierarchy of the use cases in terms of size is clear and determined by the number of edges that should be generated in each use case. We conclude this experiment by observing that the GOOSE graph outsourcing time is strongly correlated to the graph size: if you take any two graphs A and B, if A has more edges than B (cf. Y axis in Figure 3.5(b)), then the time to outsource A is larger than the time to outsource B (cf. Y axis in Figure 3.5(a)). This is particularly visible for large scaling factors,

where the hierarchy of the generated graph sizes (in terms of # of edges) is strictly followed by the hierarchy of use cases in terms of graph outsourcing times.

Scalability of query evaluation. For each of the 4 use cases, for each of the scaling factors 10^3 and 10^4, we generate with gMark 200 graphs and a workload of 5 queries coupled to each graph. Hence, we have run a total number of 8000 queries, having diverse properties specified in the gMark use cases. In particular, for each use case the generated queries are unary/binary, recursive/non-recursive (i.e., contain Kleene stars or not), linear/constant (i.e., return a number of answers that depend or not on the size of the graph), and have various shapes (chain, star, cycle, star-chain). Although we were able to easily scale the GOOSE graph outsourcing up to scaling factor 10^7, for the query evaluation experiment we evaluated queries only up to 10^4 because the bottleneck of this experiment is the standard SPARQL engine. Indeed, if we simply evaluate a generated query on a generated graph of scaling factor 10^4, it may happen that this takes already up to a minute, without any GOOSE security. This limitation of current SPARQL engines, in particular for evaluating recursive queries, has been already pointed out in the literature e.g., in [BBC+17a]. Hence, we were able to benchmark GOOSE query evaluation vs standard query evaluation only on scaling factors 10^3 and 10^4, and we run 3 times each query with each system before averaging. We show our results in Figure 3.5(c). We observe that the running times depend on the use case in the sense that if a graph has more nodes, it is more likely that a query has more results hence it may take more time to enumerate all results. This is why *uniprot* and *shop* take more time than the others. If we compare the relative performance of standard SPARQL evaluation vs GOOSE query evaluation, we observe that the overhead due to cryptographic primitives in GOOSE is dominated by the time taken by the GOOSE SPARQL engine. We also plot the relative overhead, which obviously increases when there are more query answers to encrypt and decrypt during steps 3 and 4 in GOOSE. Hence, a large overhead in this experiment is correlated to a large share of the answers translator in the next one.

Zoom of end-to-end solution. In this last experiment, we see GOOSE as an end-to-end solution consisting of outsourcing a graph and then evaluating several queries on it. In Figure 3.5(d), we show the time shares taken by each GOOSE participant, for each of the 4 use cases, for fixed scaling factor 10^4, after summing up the times needed for graph outsourcing (cf. the first experiment) and for evaluating all 5 queries in the workload (cf. the second experiment). As expected, the SPARQL engine takes the lion's share. Moreover, the next most visible participant is the answers translator, which has to decrypt hidden answers received from the SPARQL engine, translate the answers, and re-encrypt the true answers before sending them to the data client. Without surprise, the time shares taken by the two participants outside the cloud (data owner and data client) are negligible, the bulk of the computation being outsourced to the cloud.

3.4 Conclusions

We presented an overview of GOOSE, a secure framework for outsourcing graphs and querying them with SPARQL queries defined by UCRPQ. In our DBSec research paper [CL20b], we present a detailed analysis of GOOSE, from both theoretical (correctness, complexity, security) and empirical (large-scale experimental study) points of view. Our ISWC demo paper [CL20a] provides details on how to use our GOOSE open source code to reproduce the running example, as well as the experimental study from the research paper. I was in charge of the entire research process from designing the problem setting and the workflow of GOOSE, performing the theo-

retical analysis, to designing and implementing the empirical evaluation. My co-author is Pascal Lafourcade (LIMOS), who provided his expertise on the security issues.

As future work, we plan to extend GOOSE to support other practical SPARQL features such as UCRPQ where some of the query nodes are constants. Another idea would be to combine UCRPQ with aggregates. The main reason why we have not already studied these extensions is that we are not currently able to perform a large-scale experimental study of these new features, hence a potential contribution would be mostly theoretical. Indeed, although gMark was of great help for the empirical evaluation of the current version of GOOSE, gMark does not capture yet many practical query features beyond UCRPQ. Consequently, we first need to extend gMark to be able to efficiently generate more complex queries, but while keeping the finely tuned properties, in particular the selectivity estimation. This is a non-trivial gMark extension, that need to be done before attacking extensions of GOOSE.

An orthogonal direction of future work would be to investigate another possible graph query engines on which we could rely as a black-box in GOOSE. This idea is based on the observation that in the current empirical evaluation, the overhead due to cryptographic primitives is dominated by the time taken by the graph query engine. This observation is not surprising seen the first gMark-based empirical study of graph query engines [BBC+17a]. However, there are very recent promising developments in the database community [JGGL20] that could help improving the current graph query engine of GOOSE. Hence, it may be worthwhile to perform a novel in-depth study of current query engines that support UCRPQ in order to identify the best one and plug it as a block box in a future version of GOOSE.

Chapter 4

Secure Protocols for Multi-Armed Bandits

The multi-armed bandit is a standard model for decision making under uncertainty, where a learning agent repeatedly chooses an action (pull a bandit arm) and the environment responds with a stochastic outcome (reward) coming from an unknown distribution associated with the chosen arm. Bandits have a wide-range of application such as Web advertisement. We addressed the security concerns that occur when bandit data and computations are outsourced to an honest-but-curious cloud. Our approach consists of developing distributed and secure protocols that combine state-of-the-art bandit algorithms with cryptographic schemes and secure multi-party computations. We proposed such protocols for three bandit algorithms for *cumulative reward maximization* (UCB and LinUCB) and *best arm identification* (Successive Rejects). Our protocols guarantee the same result as the standard bandit algorithms, while satisfying desirable security properties such as (i) cloud nodes cannot learn the algorithm output or more than limited pieces of input, and (ii) by analyzing messages exchanged among cloud nodes, an external observer cannot learn the input or output. For each protocol, the cryptographic overhead is linear in the input size, and our implementation confirms the linear-time behavior and the feasibility of our approach. In this chapter, we present our contributions on securing the UCB algorithm, and we give a brief overview of the other protocols.

Relevant Publications

- Research papers: TrustCom [CLLS20], ProvSec [CDLS20], ISPEC [CLLS19]

Contents

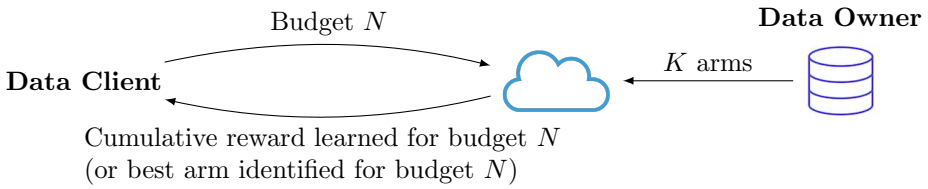


Figure 4.1: Outsourcing bandit data and algorithms.

4.1 Context

In stochastic multi-armed bandits, a learning agent sequentially needs to decide which *arm* (action/option) to pull from K arms with unknown associated values available in the learning environment. After each pull, the environment responds with a feedback, in the form of a stochastic *reward* from an unknown distribution associated with the arm chosen by the agent. This is a dynamic research topic with a wide range of applications, including clinical trials for deciding on the best treatment to give to a patient [Tho33], on-line advertisements and recommender systems [LCLS10], and game playing [Mun14].

There are two popular objectives in bandit problems and for each of them the machine learning community proposed numerous algorithms [LS20]: the most popular objective is *cumulative reward maximization* (i.e., maximize the sum of observed rewards) and the other one is *best arm identification*. In this chapter, we focus mainly on cumulative reward maximization. For this bandit objective, the learning agent has to continuously face the so-called exploration-exploitation dilemma and decide whether to *explore* by choosing arms with more uncertain associated values, or to *exploit* the information already acquired by choosing the arm with the seemingly largest associated value. We address the security concerns that occur when outsourcing the cumulative reward maximization data and computations to the cloud. Our scenario is inspired by the *machine learning as a service* cloud computing model, for which security is known as a major concern [BMMP18]. As a motivating example, assume:

• A *data owner*: a company that wants to monetize some collected data, while keeping ownership over it. The collected data may be a large quantity of surveys on customer preferences for several products. By product, we mean any type of object or service. The K bandit arms are the surveyed products and only the data owner knows their associated rewards, based on the collected surveys.

• A *data client*: a company that wants to spend some budget to use some of the data owner's data. The data client may be a small company that cannot afford doing its own surveys, but wants to estimate the income that it could generate for the products surveyed by the data owner. The cumulative reward captures such information because it sums the rewards produced by each product. The budget N is the number of data owner's surveys used to compute the cumulative reward and the bandit algorithm has to decide *how* to choose these N surveys in order to maximize the cumulative reward. A larger budget gives a higher accuracy for the largest cumulative reward. The data client only sees the cumulative reward, without knowing the values associated to each arm.

We assume that the interaction between the data owner and the data client is done using the *cloud* (cf. Figure 4.1), where both data and computations are *outsourced*. The data owner does the data outsourcing, and the data client interacts directly with the cloud, by sending the budget and receiving the obtained cumulative reward. The outsourced data may be sensitive (e.g., personal, commercial, or medical data). We want the outsourced learning algorithm to be run while protecting data against unauthorized access. The problem that we address is how to

allow the data client to obtain precisely the same cumulative reward as with a standard bandit algorithm, *Upper Confidence Bound (UCB)* [ACF02], within a reasonable computation time and while preserving the data security. Indeed, the outsourced data can be communicated over an untrustworthy network and processed on some untrustworthy machines, where a curious cloud admin may learn sensitive data that belongs only to the data owner.

Enhancing bandit algorithms with security/privacy guarantees is not a trivial problem. For instance, [GUK18, MT15, TD16] use *differential privacy* [Dwo06]. However, in these approaches, the returned reward is not the same reward obtained for the data client's budget using standard algorithms. This happens because differential privacy guarantees depend on noise being injected in the input/output. We take a complementary approach by relying on *cryptography*, which is to the best of our knowledge an original approach. Our goal is to have security guarantees while obtaining the same output as standard (non-secure) algorithms. The security for obtaining the same output has a price because the computation time increases because of cryptographic primitives that are time-consuming in practice. More precisely, we require that the data owner (which can be seen as an oracle knowing the reward functions associated with each arm) encrypts her data before outsourcing it to the cloud. Then, the (encrypted) output of the cumulative reward maximization algorithm should be exactly the same as for standard UCB, with the cost of an increased computation time. From a theoretical viewpoint, the problem can be straightforwardly solved by relying on a fully homomorphic encryption scheme [Gen09], which allows to compute any function directly in the encrypted domain. However, it remains an open question how to make such a scheme work fast and be accurate in practice. Indeed, the state-of-the art fully homomorphic systems (SEAL [SEA] and HElib [HEl]) yield only approximate results when they work with real numbers, by using the CKKS scheme [CKKS17]. Hence, it is not currently possible to program an algorithm such as UCB in a fully homomorphic system and obtain exactly the same result as standard UCB.

Consequently, our approach is to rely on *simpler cryptographic schemes in conjunction with secure multi-party computations* i.e., design a distributed protocol with several cloud node participants such that each of them can only learn the specific data needed for performing its task and nothing else. For instance, if a participant does in clear computations on real numbers, these computations concern data of only one arm, and no other participant has access to this piece of data. We assume the same *honest-but-curious* cloud model as in Section 3.3.2, following the classical formulation in [Gol04, Chapter 7]. The data client indicates to the cloud her budget N and receives the cumulative reward R that the cloud computes using the K arms outsourced by the data owner and the data client's budget N. The data client does not have to do any computation, except for decrypting R when the data client receives this information encrypted from the cloud. We expect the following *security properties*:

1. No cloud node can learn the cumulative reward.
2. The data client cannot learn information about the rewards produced by each arm or which arm has been pulled at some round.
3. By analyzing the messages exchanged between different cloud nodes, an external observer cannot learn the cumulative reward, the sum of rewards produced by some arm, or which arm has been pulled at some round.

We give a brief intuition for each property. Property 1 implies that only the data client can see in clear the cumulative reward for which she spends a budget. Property 2 ensures that the data client can see only the information for which she pays, and nothing else. Otherwise, depending on the difficulty of the bandit problem, the data client could estimate the arm values based on the contribution of each arm to the cumulative reward, which would leak information that should be known only by the data owner. Property 3 states that if some curious cloud admin analyzes

all messages exchanged over the network, then she should not learn the input or output of the cumulative reward maximization algorithm.

Our protocol UCB-DS satisfies these properties by exchanging only encrypted messages, and by distributing the computations among several cloud node participants, each of them having access only to the specific data that it needs for performing its task and nothing else. The challenge is to efficiently distribute tasks among as few cloud participants as possible, while minimizing the time needed for cryptographic primitives. We stress that our proposed protocol returns exactly the same cumulative reward as UCB. To achieve our goals, we rely on *indistinguishable under chosen-plaintext attack* (IND-CPA) cryptographic schemes: symmetric encryption AES-CBC [AES01, BDJR97] and asymmetric partially homomorphic Paillier's scheme [Pai99].

4.2 Related Work

We first give some background information on the classical UCB algorithm. Then, we discuss existing works on enhancing bandit algorithms with security/privacy guarantees.

Upper Confidence Bound (UCB) is a class of algorithms commonly used when facing the exploration-exploitation dilemma. Each bandit arm is associated with a distribution whose mean is unknown to the learning agent. When pulling an arm, the agent observes an independent and identically distributed reward drawn from the distribution associated to the chosen arm. For instance, if the rewards are drawn from Bernoulli distributions with expected values $\mu_1, \ldots, \mu_K$ unknown to the agent, a call to the function $pull(i)$ for a chosen arm i randomly returns 0 or 1 according to the associated Bernoulli distribution, i.e., the probability of returning 1 is μ_i and the probability of 0 is 1–μ_i. The agent sequentially selects the N arms to be pulled with the goal of maximizing the sum of rewards. To guide the choice of the learner, arm scores have been proposed to construct *upper confidence bounds* (UCB) based on the empirical mean of arm-specific rewards and the number of arm pulls. Specifically, in the UCB algorithm [ACF02] presented in Figure 4.2, for each arm i, the score B_i is an upper-confidence bound on μ_i, obtained as the sum between (i) the *exploitation* term given by the empirical mean of rewards observed from arm i, and (ii) the *exploration* term, which takes into account the uncertainty. Notice that after each observed reward, scores for all arms are updated, since the *exploration* term $\sqrt{\frac{2\ln(t)}{n_i}}$ depends on the total number of rewards observed up to current round t. Thus, an arm i being pulled few times (i.e., with small n_i) will have a relatively large *exploration* term. The score B_i is thus an *optimistic* estimate for the value associated to arm i, since it can be interpreted as the largest statistically plausible mean value associated to arm i, given the observed rewards. As shown in Figure 4.2, UCB chooses to pull next the arm with the largest updated B_i score, thus following the principle of *optimism in the face of uncertainty*. This principle suggests to follow what seems to be the best arm, based on the *optimistically* constructed scores. The same principle is employed in various sequential decision making problems (see [Mun14] for a survey).

Bandit algorithms with security/privacy guarantees. Each line in Table 4.1 corresponds to a standard problem in stochastic multi-armed bandits. As already mentioned, the most popular problem is *cumulative reward maximization* and UCB is a standard algorithm for solving it [ACF02]. There is a recent line of research on enhancing algorithms such as UCB with differential privacy [GUK18, MT15, TD16]. There are some fundamental differences between this line of work and our work based on cryptography. On the one side, the running time overhead of differentially-private algorithms is negligible, whereas our approach has an overhead in computation time coming from the use of cryptographic primitives. On the other side, the cumulative

Input: Budget N, number of arms K

Unknown environment: K distributions associated to the K arms, with expected values $\mu_1, \ldots, \mu_K$ unknown to the learning agent. The agent has access to a reward function $pull(.)$ that can be called N times. A call $pull(i)$ returns a random value from the distribution associated to arm i.

Output: Sum of observed rewards for all arms

/ Initialization: pull each arm once & initialize variables */*
for $1 \leq i \leq K$
  let $r = pull(i)$ */* Random reward for arm i */*
  let $s_i = r$ */* Sum of observed rewards for arm i */*
  let $n_i = 1$ */* Number of pulls of arm i */*
  let $B_i = \frac{s_i}{n_i} + \sqrt{\frac{2\ln(K)}{n_i}}$ */* B_i is an UCB on μ_i */*
/ Exploration-exploitation: pull one arm at each round t */*
for $K+1 \leq t \leq N$ */* Only a budget of $N-K$ is left */*
  let $i_m = arg\,max_{1 \leq i \leq K}(B_i)$ */* Ties broken randomly */*
  let $r = pull(i_m)$
  let $s_{i_m} = s_{i_m} + r$
  let $n_{i_m} = n_{i_m} + 1$
  for $1 \leq i \leq K$
    let $B_i = \frac{s_i}{n_i} + \sqrt{\frac{2\ln(t)}{n_i}}$
return $s_1 + \ldots + s_K$

Figure 4.2: UCB Algorithm [ACF02].

reward returned by differentially-private algorithms is different from the output of standard UCB. Indeed, to obtain differentially-private guarantees for a bandit algorithm, noise is added to the algorithm input or output. Thus, the cumulative reward obtained using a differentially-private algorithm is different from that obtained by the algorithm without privacy guarantees. This is reflected in the *regret analysis* of the algorithms (where the *regret* is given by the difference in the cumulative reward obtained by a learning agent and the best cumulative reward possible obtained by always playing the best arm): the regret of differentially-private bandit algorithms have as overhead an additive [TD16] or multiplicative factor [GUK18, MT15] with respect to the regret of their non-private version. In contrast, our cryptography-based protocol UCB-DS [CLLS20] is guaranteed to return exactly the same cumulative reward as standard UCB.

All related works discussed thus far are for standard stochastic bandits, where the arms are independent i.e., observing a reward for an arm gives no information about the rewards associated to the other arms. The classical UCB algorithm has been applied to *linear bandits* [APS11, Aue02, LCLS10], where the arms are fixed vectors and the rewards are given by an unknown linear function, common to all arms. Following [LS20, Chapter 19], we use LinUCB as a generic name for UCB applied to linear bandits and we specifically rely on the algorithm in [APS11]. There already exists a differentially-private version [SS18] of LinUCB, for which the regret is different compared to the standard version. We proposed a cryptography-based protocol [CDLS20] that returns exactly the same result as LinUCB.

The second line in Table 4.1 corresponds to a different bandit problem that is *best arm identification*, equivalent to minimizing the simple regret, that is the difference between the values associated with the arm that is actually the best and the best arm identified by the algorithm. This problem has been extensively studied in the machine learning community [ABM10, GGL12,

Table 4.1: Summary of related work and positioning of our contributions (in **bold**).

	Differential privacy	**Cryptography**
Cumulative reward maximization aka cumulative regret minimization	[GUK18, MT15, TD16, SS18]	**[CLLS20, CDLS20]**
Best arm identification aka simple regret minimization	Not yet studied to the best of our knowledge	**[CLLS19]**

KCG16, SLM14], but to the best of our knowledge, the problem has not been addressed from a privacy/security viewpoint. We proposed a cryptography-based protocol [CLLS19] that enhances the Successive rejects algorithm [ABM10] for best arm identification with security guarantees, while obtaining the same result as Successive rejects.

As a conclusion of the related work, to the best of our knowledge, our line of research is the first that adds security guarantees to bandit algorithms using cryptographic techniques. We also stress that the three protocols that we proposed [CLLS20, CDLS20, CLLS19] are different and cannot be reduced to one another because the underlying algorithms are different in terms of bandit objective (cumulative reward maximization [CLLS20, CDLS20] vs best arm identification [CLLS19]) and bandit model (standard stochastic [CLLS19, CLLS20] vs linear [CDLS20]).

4.3 Summary of Contributions

In Section 4.3.1, we present UCB-DS (Distributed and Secure protocol based on the UCB algorithm). In Section 4.3.2, we briefly discuss the theoretical analysis of UCB-DS in terms of security, correctness and complexity, as well as the UCB-DS2 refinement. In Section 4.3.3, we include a proof-of-concept empirical evaluation that confirms the theoretical complexity, and shows the scalability and feasibility of our protocols. In Section 4.3.4, we summarize our secure and distributed protocols for other bandit problems.

4.3.1 Worfklow of UCB-DS

We first introduce the cryptographic tools on which relies the UCB-DS protocol. Then, we give an overview of the protocol's workflow, and discuss each of its steps.

Cryptographic tools. We briefly introduce two cryptographic schemes on which we rely: Paillier and AES-CBC, which are both IND-CPA secure.

- *Paillier asymmetric encryption.* Paillier's cryptosystem is *additive homomorphic* [Pai99]. Let m_1 and m_2 be two plaintexts. The product of the two associated ciphertexts with the public key pk, denoted $c_1 = \mathcal{E}_{\mathsf{pk}}(m_1)$ and $c_2 = \mathcal{E}_{\mathsf{pk}}(m_2)$, is the encryption of the sum of m_1 and m_2. Indeed, we have: $\mathcal{E}_{\mathsf{pk}}(m_1) \cdot \mathcal{E}_{\mathsf{pk}}(m_2) = \mathcal{E}_{\mathsf{pk}}(m_1 + m_2)$. We also denote by $\mathcal{D}_{\mathsf{sk}}(c)$ the decryption of the cipher c by the secret key sk.
- *AES-CBC symmetric encryption.* AES [AES01] is a NIST standard for encrypting messages of 128 bits. We use it with CBC mode (Cipher Block Chaining) and denote $c = \mathsf{Enc}(m)$ the encryption of m and $m = \mathsf{Dec}(c)$ the decryption of c with the same symmetric key shared between the participants.

Both Paillier and AES-CBC are *IND-CPA*: (i) Paillier is IND-CPA under the decisional composite residuosity assumption [Pai99], and (ii) AES-CBC is IND-CPA under the assumption that AES is a pseudo-random permutation [BDJR97]. All theoretical security properties of our protocols also hold if we choose any other IND-CPA symmetric scheme instead of AES-CBC, and any

other additive homomorphic IND-CPA asymmetric scheme instead of Paillier. Our choice to rely on the aforementioned schemes is due to practical reasons. AES-CBC is very efficient in practice and implemented in standard libraries for modern programming languages. Paillier is also supported by a number of libraries that can be used in practice.

Overview of UCB-DS. In Figure 4.3, we present an overview of UCB-DS. There are $K+1$ cloud participants: K arm nodes R_i and a node AS (Arm Selector) that is the controller of the protocol. We assume that the data owner and all cloud participants share the same symmetric AES-CBC key, used for encryption function Enc. The data client (DC) generates a Paillier's key pair (pk, sk) and for sake of clarity we denote $\mathcal{E}_{\mathsf{DC}}(m)$ for $\mathcal{E}_{\mathsf{pk}}(m)$. By $[\![x]\!]$, we denote the set $\{1,\ldots,x\}$, and by $y||z$ we denote the concatenation of y and z. UCB-DS works as follows:

- Figure 4.3(a) (steps 0 and 1). For $i \in [\![K]\!]$, the data owner outsources to arm node R_i the reward function (encrypted with Enc) associated to arm i. The data client sends to the cloud her budget N.
- Figure 4.3(b) (steps 2, 3, and 4). This is the core of the protocol, being done during $1+N-K$ iterations: once for the initialization phase of UCB and $N-K$ times for the exploration-exploitation phase of UCB cf. Figure 4.2. For each iteration, all arm nodes interact to decide which arm should be pulled next and communicate this information to AS. The arm nodes communicate in a random order, which changes at each iteration. All messages exchanged between nodes are encrypted with Enc. Although each arm node stores information about its rewards, it never reveals this information to other nodes.
- Figure 4.3(c) (steps 5 and 6). After spending the data client's budget, each arm node sends to AS the sum of rewards that it produced, encrypted with $\mathcal{E}_{\mathsf{DC}}$. Due to the additive homomorphic property of Paillier cryptosystem, AS is able to sum up the K partial rewards to compute the cumulative reward $\mathcal{E}_{\mathsf{DC}}(R)$ directly in the encrypted domain. Only the data client can decrypt this information.

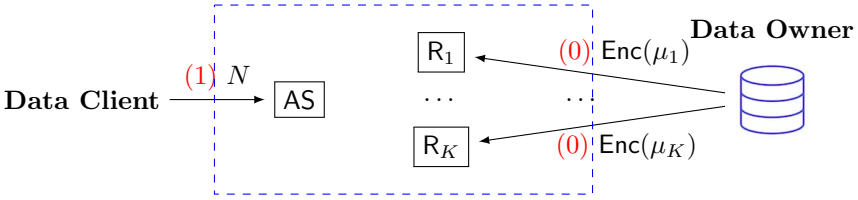
We next detail each step and present pseudocode only when the step is not trivial.

Step 0. We recall (cf. Figure 4.2) that the data owner knows $\mu_1,\ldots,\mu_K$ defining K Bernoulli distributions associated to the K arms. The data owner sends to each arm node R_i the encrypted value $\mathsf{Enc}(\mu_i)$, for $i \in [\![K]\!]$. Since the data owner and the cloud share the symmetric key, then each arm node R_i can decrypt and obtain μ_i. Moreover, each node R_i initializes to 0 the following two variables that it later on updates during the protocol: s_i (i.e., sum of rewards for arm i) and n_i (i.e., number of times the arm i has been pulled). Additionally, each arm node R_i initializes a variable $t = K-1$, which is later on updated and needed for the computation of B_i.

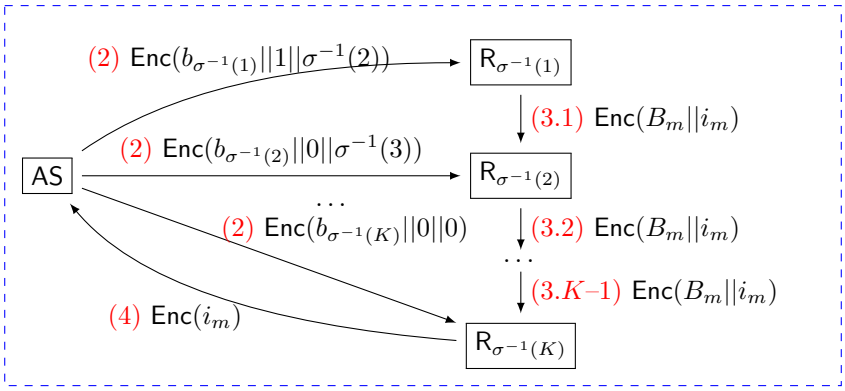
Step 1. The data client sends her budget N to AS.

Pseudocodes of Steps 2, 3, and 4 are presented in Figure 4.4.

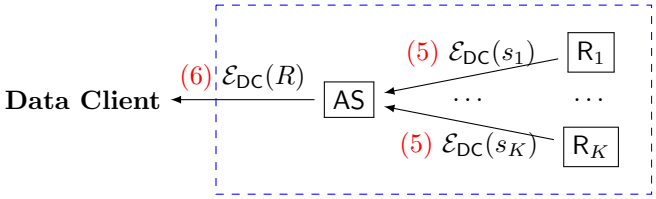
Step 2. It corresponds to everything except the last two lines in Figure 4.4(a) and has $1+N-K$ iterations. At each iteration, AS sends to the R_i nodes a bit b_i indicating whether the arm i should be pulled or not. At the first iteration (that corresponds to the initialization phase of UCB cf. Figure 4.2), AS sends $b_i = 1$ to each arm, and at the next $N-K$ iterations (that correspond to the exploration-exploitation phase of UCB cf. Figure 4.2), AS sends $b_i = 1$ only to a chosen arm i_m and sends $b_i = 0$ to all other arms. Moreover, at each iteration, AS generates a permutation $\sigma : [\![K]\!] \rightarrow [\![K]\!]$ (i.e., a function for which every element occurs exactly once as an image value), based on which AS computes two more components that it sends to R_i: first_i that indicates whether the arm node is the first of the ring hence it should initialize B_m and i_m, and



(a) Steps 0 and 1 are done only once at the beginning.



(b) Steps 2, 3, and 4 are done $1 + N - K$ times.



(c) Steps 5 and 6 are done only once at the end.

Figure 4.3: Overview of UCB-DS. The dashed rectangle is the cloud.

$next_i$ that indicates to which node the updated B_m and i_m should be sent during Step 3. The arm node that receives 0 on the *next* component is the last one of the ring and sends i_m to AS, which thus knows which arm should be pulled next. Each information that AS sends to R_i is thus useful for the ring computation of i_m in Step 3.

Step 3. This step corresponds to everything except the last two lines in Figure 4.4(b). Note that the variable t stores how many arm pulls have been done in total since the beginning of the protocol. As discussed for Step 0, each arm initialized $t = K - 1$, hence $t = K$ after the first iteration of AS, which allows to compute the first B_i values at the end of the initialization phase. Then, during the next $N - K$ iterations of AS, the variable t is incremented, which allows to compute B_i values during the exploration-exploitation phase. To decide which arm has the highest B_i and should be pulled at the next iteration, the arm nodes R_i do a distributed ring computation, where the first arm node according to permutation σ (i.e., the only arm node that received $first_i$=1) initializes max value B_m and argmax i_m. At each ring iteration (Steps 3.1, ..., 3.K-1, cf. Figure 4.3(b)), the current arm node sends updated B_m and i_m to the next arm node cf. σ. Even though B_m and i_m do not change, it is important to re-encrypt $\mathsf{Enc}(B_m||i_m)$

let $i_m = 0$
for $j \in [\![N - K + 1]\!]$
  let σ=random permutation of $[\![K]\!]$
  for $i \in [\![K]\!]$
    / b_i is a bit indicating if arm i should be pulled */*
    if $i_m = 0$ **or** $i_m = i$ **then let** $b_i = 1$ **else let** $b_i = 0$
    / $first_i$ is a bit indicating if i is the first arm node in the ring cf. σ */*
    if $\sigma(i) = 1$ **then let** $first_i = 1$ **else let** $first_i = 0$
    / $next_i$ indicates the next arm node in the ring, or 0 if i is the last cf. σ */*
    if $\sigma(i) \neq K$ **then let** $next_i = \sigma^{-1}(\sigma(i) + 1)$ **else let** $next_i = 0$
    send $\mathsf{Enc}(b_i||first_i||next_i)$ to arm node R_i
  receive ciphertext from arm node $\mathsf{R}_{\sigma^{-1}(K)}$
  / ciphertext is* $\mathsf{Enc}(i_m)$ **/*
  let $i_m = \mathsf{Dec}$(ciphertext)

(a) Pseudocode of AS.

receive ciphertext1 from AS
/ ciphertext1 is* $\mathsf{Enc}(b_i||first_i||next_i)$ **/*
let $b_i||first_i||next_i = \mathsf{Dec}$(ciphertext1)
let $t = t + 1$
if $b_i = 1$ */* Pull arm i and update its variables */*
  let $r = pull(i)$
  let $s_i = s_i + r$
  let $n_i = n_i + 1$
let $B_i = \frac{s_i}{n_i} + \sqrt{\frac{2\ln(t)}{n_i}}$
if $first_i = 0$
  receive ciphertext2 from preceding arm node in ring
  / ciphertext2 is* $\mathsf{Enc}(B_m||i_m)$ **/*
  $B_m||i_m = \mathsf{Dec}$(ciphertext2)
if $first_i = 1$ **or** $B_m < B_i$
  let $i_m = i$
  let $B_m = B_i$
if $next_i \neq 0$
  send $\mathsf{Enc}(B_m||i_m)$ to R_{next_i}
else
  send $\mathsf{Enc}(i_m)$ to AS

(b) Pseudocode of R_i, for $i \in [\![K]\!]$.

Figure 4.4: Pseudocode of AS and R_i during steps 2, 3, and 4 cf. Figure 4.3(b).

before sending it to the next node to prevent an external observer from knowing when there is a change in the max and argmax (and hence learn information about which arms are pulled more often). Finally, once the ring computation reaches the last arm node relative to σ (i.e., the only one that received $next_i = 0$), we go to Step 4.

Step 4. This step corresponds to the last two lines in Figure 4.4(b) (the last arm node in the ring sends $\mathsf{Enc}(i_m)$ to AS), followed by the last two lines in Figure 4.4(a) (AS receives and decrypts the index of the arm to be pulled at the next iteration).

Step 5. Once the budget is spent and no more arm has to be pulled, each arm node R_i (for $i \in [\![K]\!]$) encrypts with $\mathcal{E}_{\mathsf{DC}}$ its sum of rewards s_i and sends the result $\mathcal{E}_{\mathsf{DC}}(s_i)$ to AS.

Step 6. The node AS takes the K ciphertexts $\mathcal{E}_{\mathsf{DC}}(s_i)$ received at Step 5, and computes $\mathcal{E}_{\mathsf{DC}}(R) = \mathcal{E}_{\mathsf{DC}}(\sum_{i=1}^{K} s_i) = \prod_{i=1}^{K}(\mathcal{E}_{\mathsf{DC}}(s_i))$, thanks to the additive homomorphic property of Paillier cryptosystem. Then, AS sends $\mathcal{E}_{\mathsf{DC}}(R)$ to the data client, who is able to decrypt using her sk and hence obtains R.

4.3.2 Theoretical Analysis of UCB-DS

The main results outlined in this section are:

- (Security) We characterize the security properties satisfied by UCB-DS.
- (Complexity) We quantify the number of cryptographic primitives in UCB-DS: $O(NK)$ AES-CBC encryptions/decryptions, K Paillier encryptions, and one Paillier decryption.
- (Correctness) We point out that UCB-DS returns the same result as standard UCB.
- (Refinement) We propose UCB-DS2, with stronger security guarantees at the price of K more AES-CBC keys and $O(NK)$ more AES-CBC encryptions/decryptions, and the same number of Paillier encryptions/decryptions.

Security. In Table 4.2, we summarize what each participant in UCB-DS knows/does not know. The main properties of our protocol are:

- No cloud node can learn the cumulative reward and additionally:
  - Only AS and the pulled arm know which arm is pulled at each round. Arms that are not pulled can guess the pulled arm with average probability $\frac{1}{2} + \frac{1}{2K}$.
  - Only arm node R_i knows the sum of rewards for arm i.
- Only DC knows the cumulative reward, and she knows nothing else.
- An external observer cannot learn the cumulative reward, the sum of rewards for some arm, or which arm has been pulled at some round.

These properties subsume the list of desirable security properties listed in Section 4.1. We include in [CLLS20] the formal statements and proofs for all these security properties, which rely on the IND-CPA security of AES-CBC and Paillier.

Complexity. We detail in [CLLS20] the number of cryptographic operations used in each step of UCB-DS. By summing up, we obtain $O(NK)$ AES-CBC encryptions/decryptions, K Paillier encryptions, and one Paillier decryption. Hence, we have a number of AES-CBC operations linear in N, whereas the number of Paillier operations does not depend on N. These are desirable complexity properties. In particular, the number of Paillier operations (which are quite slow to evaluate in practice) depends only on K that is typically much smaller than N in bandit scenarios. Our implementation (cf. Section 4.3.3) follows the aforementioned theoretical analysis and confirms the linear time behavior and the scalability of UCB-DS.

Correctness. In [CLLS20], we point out a reduction from UCB-DS to standard UCB. The reduction relies on the consistency properties of AES-CBC and Paillier. In particular, the random permutation σ (that is generated at each round to decide in which order to iterate over arms)

Table 4.2: What each participant of UCB-DS knows and does not know.

Participant	*Knows*	*Does not know*
AS	• Arm pulled at each round	• Sum of rewards for some arm and cumulative reward
R_i	• Sum of rewards for arm i • Arm pulled at each round, with average probability $\frac{1}{2} + \frac{1}{2K}$	• Sum of rewards of other arm $j \neq i$ and cumulative reward
DC	• Cumulative reward	• Arm pulled at each round • Sum of rewards for some arm
External observer	• Nothing	• Arm pulled at each round • Sum of rewards for some arm and cumulative reward

reduces to the randomness in the argmax function used in UCB cf. Figure 4.2 when, if several arms have maximal B_i-value, then the argmax should be randomly picked among those arms.

Refinement. The UCB-DS2 refinement adds slightly stronger security guarantees to UCB-DS, for few more cryptographic operations (but the similar asymptotic behavior as UCB-DS). A property of UCB-DS (cf. Table 4.2) is that an arm node R_i knows with average probability of $\frac{1}{2}+\frac{1}{2K}$ what arm is pulled at the next round. This happens because during the ring computation, every arm sees in clear the partial argmax i_m. Our UCB-DS2 refinement removes this leakage. The idea of UCB-DS2 is that, in addition to UCB-DS, we also encrypt the partial argmax i_m during the ring computation. This modification requires to introduce new keys. We recall that UCB-DS assumes an AES-CBC key that is shared between the data owner and all cloud participants and that is used for the functions Enc/Dec. For UCB-DS2, if we want that an arm node R_i cannot decrypt the partial argmax i_m received from the previous arm node in the ring, we need to encrypt i_m with some other key. This is why in UCB-DS2 we introduce K new AES-CBC keys, each of them shared between AS and a single R_i arm node. Each such key defines functions $\mathsf{Enc}_i/\mathsf{Dec}_i$. We include in [CLLS20] the pseudocode and the analysis of UCB-DS2.

4.3.3 Empirical Evaluation of UCB-DS

We show that the overhead due to cryptographic primitives is reasonable, hence our protocols are feasible. More precisely, we show the scalability of our protocols with respect to both parameters N and K through an experimental study using synthetic and real data. We compare the standard UCB [ACF02] (outlined in Figure 4.2) with three distributed protocols:

- UCB-D = Distributed UCB in the spirit of UCB-DS cf. Section 4.3.1, but with all messages exchanged in clear among participants (i.e., UCB-D does not use any cryptographic primitive). The only overhead with respect to UCB is due to distribution of tasks.
- UCB-DS = Distributed Secure UCB cf. Section 4.3.1.
- UCB-DS2 = Refinement of UCB-DS cf. Section 4.3.2.

We implemented the algorithms in Python 3. For AES-CBC we used the *PyCryptodome* library[1] and keys of 256 bits. For Paillier, we used the *phe* library[2] in the default configuration with keys of 2048 bits. We did our experiments on a laptop with CPU Intel Core i7 of 2.80GHz and 16GB of RAM, running Ubuntu. Each reported result is averaged over 100 runs. In each run,

[1] https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html
[2] https://python-paillier.readthedocs.io/en/develop/

we executed all algorithms using the same random seeds, needed for drawing arm rewards and for generating the permutation used to iterate in a random order over the arms when choosing the argmax arm to be pulled at the next round.

We make available on a public GitHub repository[3] our source code, together with the data that we used, the generated results from which we obtained our plots, and scripts that allow to install the needed libraries and reproduce our plots.

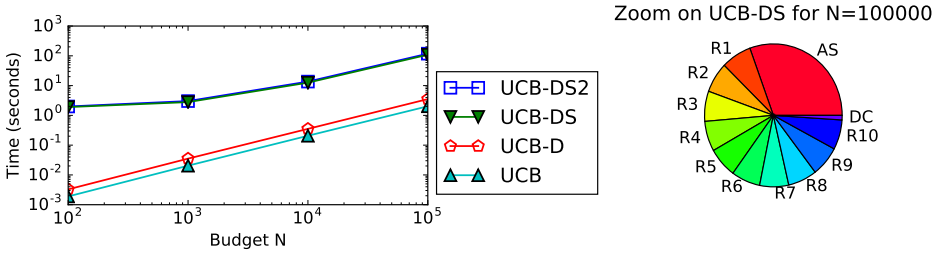
As expected, in each experiment, all four algorithms output exactly the same cumulative reward. The property that our secure algorithms return exactly the same cumulative reward as standard UCB is in contrast with differentially-private multi-armed bandit algorithms [GUK18, MT15, TD16], where the returned cumulative rewards are different from that of standard UCB. Consequently, a shallow empirical comparison between these works and ours boils down to comparing apples and oranges: (i) on the one hand, the running time of differentially-private bandit algorithms is roughly the same as for standard UCB and is never reported in their experiments, whereas (ii) on the other hand, for our algorithms the cumulative reward is always the same as for standard UCB and consequently there is no point for us in doing any plot on the cumulative reward. Nevertheless, we carefully analyzed all experimental settings (N, K, μ) used in the related work, that we adapt for our scalability experiments, as we detail next.

Scalability with respect to N. We rely on scenarios from the related work [GUK18, TD16] to fix K and μ, and to vary N. In Figure 4.5(a), we show the results only for Scenario 1 [GUK18]. We omit here the other scenarios, which yield similar results, included in [CLLS20]. We vary N from 10^2 to 10^5 that is also the maximum budget from [GUK18, TD16]. UCB and UCB-D have very close running times, and up to two orders of magnitude smaller than UCB-DS and UCB-DS2, which are also very close. All algorithms have a similar linear time behavior. The overhead between secure and non-secure algorithms comes naturally from the cryptographic primitives. Moreover, the two lines corresponding to the secure algorithms are not parallel with the other two lines because, cf. Section 4.3.2, the overhead due to Paillier encryptions depends only on K (that is fixed in the figure) and not on N (that varies in the figure), hence the Paillier overhead is more visible for small N. The running times of UCB-DS/UCB-DS2 for the largest considered budget $N{=}10^5$ is of $\sim$100 seconds, which remains practical. In Figure 4.5(a), we also zoom on the time taken by each participant of UCB-DS for $N{=}10^5$. We observe that AS takes the lion's share, which is expected because at each round AS sends encrypted messages to all R_i participants, whereas each R_i sends an encrypted message only to one other participant. As expected, all R_i take roughly the same time. The shares of the data owner and the data client are the smallest among all participants. This is a desirable property because we require them to do as few computations as possible, the bulk of the computation being outsourced to the cloud.

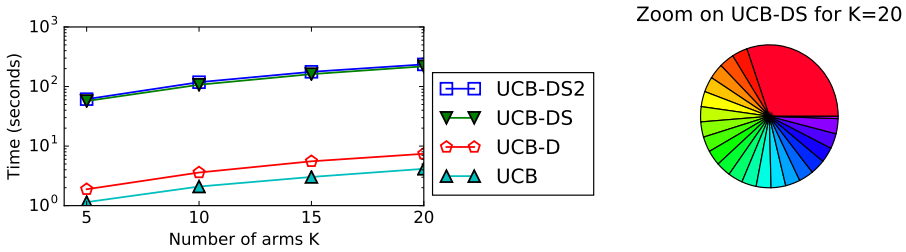
Scalability with respect to K. We fix $N{=}10^5$, and vary $K{\in}\{5, 10, 15, 20\}$ and implicitly μ with $\mu_1{=}0.9$ and $\mu_{2\leq i\leq K}{=}0.8$. We present results in Figure 4.5(b). We observe, as in the previous experiment, a linear time behavior and a similar zoom on the time taken by each participant.

Real-world data. We also stress-tested our algorithms on real-world data, using the same data and setup as [KSS13]. After a pre-processing step (detailed in [CLLS20]), we transformed real-world data in three bandit scenarios: Jester-small ($K{=}10$) and Jester-large ($K{=}100$) based on [GRGP01], and MovieLens ($K{=}100$) based on [HK16]. We ran each of these scenarios with $N{=}10^5$ that is the largest budget considered in [KSS13]. Our results (cf. Figure 4.5(c)) essentially confirm the behavior observed in the synthetic experiments i.e., there are roughly two orders of

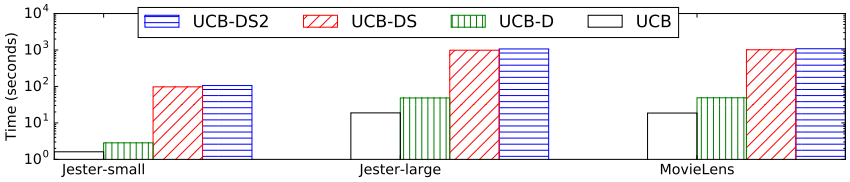
[3] https://github.com/radu1/secure-ucb



(a) Scalability with respect to N, for fixed $K = 10$, $\mu_1 = 0.9$, $\mu_2 = \ldots = \mu_{10} = 0.8$. In the zoom, we do not show DO (its share is close to 0).



(b) Scalability with respect to K, for fixed $N = 10^5$. In the zoom (labels not shown because they would be colliding): the AS takes the lion's share, $\mathsf{R}_{1 \leq i \leq 20}$ take the 20 equal shares, DC is barely visible, and DO is not shown since its share is close to 0.



(c) Running times on three real-world data scenarios from [KSS13].

Figure 4.5: Experimental results on UCB-DS.

magnitude between non-secure and secure algorithms. In the largest considered scenarios (Jester-large and MovieLens, both with K=100), where standard UCB takes around twenty seconds, both UCB-DS and UCB-DS2 take around one thousand seconds, that we believe acceptable as waiting time for the data client before getting the cumulative reward result for which she pays.

4.3.4 Other Protocols

Secure cumulative reward maximization in linear bandits. In linear bandits, the input set of arms is a fixed subset of $\mathbb{R}^d$, revealed to the learner at the beginning of the algorithm. When pulling an arm, the learner observes a noisy reward whose expected value is the inner product between the chosen arm and an unknown parameter θ characterizing the underlying linear function, common to all arms. We proposed LinUCB-DS [CDLS20], a secure and distributed protocol that returns the same cumulative reward as the LinUCB algorithm [APS11], while hiding θ from the cloud. There are two key ideas behind our protocol. (i) The data owner uses Paillier to encrypt each value of θ before outsourcing it to the cloud; the arm pulls return encrypted rewards and the cumulative reward is updated directly in the encrypted domain. (ii) The computations

are distributed between two cloud nodes (in the spirit of private outsourced sort [BO15]): one node takes care of drawing the rewards and updating the variables, without being able to decrypt the data, and another node is responsible only for comparing the estimated arm values in order to decide which arm should be pulled next; the data owner uses the public key of this second node for the data outsourcing. We refer to [CDLS20] for details on the workflow of LinUCB-DS, its theoretical and empirical analysis, and a discussion on how it can be easily adapted to secure another linear bandit algorithm, SpectralUCB [VMKK14].

Secure best arm identification in multi-armed bandits. We proposed a secure and distributed protocol [CLLS19] based on the Successive Rejects algorithm for best arm identification [ABM10]. An important difference with respect to cumulative reward maximization algorithms is that Successive Rejects allocates the user's budget in K–1 phases. After each phase, an arm is rejected. The only arm that is not rejected after the last phase is identified as being the best arm. Since the number of candidate arms diminishes throughout the phases, our protocol distributes the data and computations by phase rather than by arm. Hence, the distribution strategy is quite different compared to UCB-DS. We refer to [CLLS19] for details on the workflow of our protocol for best arm identification, and its theoretical and empirical analysis.

4.4 Conclusions

We presented an overview of our work on secure protocols for multi-armed bandits. This is an original research direction that I initiated, in close collaboration with Pascal Lafourcade (LIMOS) and Marta Soare (LIFO). We have three research papers (TrustCom [CLLS20], ProvSec [CDLS20], and ISPEC [CLLS19]), each of them on securing a different bandit algorithm. For each of these three papers, I was involved in the entire research process, from designing the problem setting and the workflow of our distributed protocols, to their empirical evaluations. The other collaborators on secure bandits were Marius Lombard-Platet (PhD student of Pascal, who participated to [CLLS20, CLLS19]), and Anatole Delabrouille (M2 student that I co-advised with Pascal and Marta, who contributed to [CDLS20]).

As future work, we plan to extend our scenario such that multiple data clients concurrently submit budgets to the cloud and receive corresponding cumulative rewards. In such a scenario, parallelism between nodes could be leveraged to improve the system throughput.

From a different point of view, we plan to investigate how far we can generalize our protocols in order to cover other bandit algorithms. For instance, an idea is to study whether the same distribution strategy from UCB-DS [CLLS20] can be used to secure other standard bandit algorithms for cumulative reward maximization, such as ε-greedy or Thompson sampling [LS20]. Another idea is to investigate if our protocol for best arm identification [CLLS19] could be generalized for other types of reward distributions. We also believe that our protocol for cumulative reward maximization in linear bandits [CDLS20] can be generalized for other bandit models.

Furthermore, seen that the research and development community on fully homomorphic encryption is actively working on improving the current systems [SEA, HEl], it may be the case that in a few years the current limitations (speed and support for real numbers) will not hold anymore. Hence, it would be interesting to perform a thorough empirical comparison between bandit algorithms implemented in fully homomorphic encryption systems vs our secure protocols.

Chapter 5

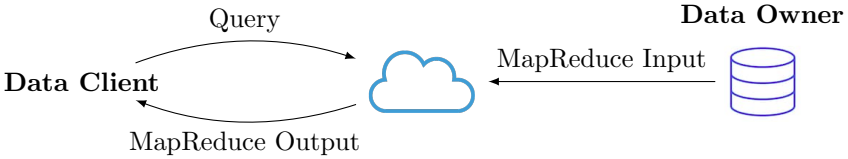
Secure MapReduce Protocols

MapReduce is one of the most popular programming paradigms that allows a user to process large data sets in parallel on a cluster. The MapReduce users often outsource their data and computations to a public cloud provider, which yields inherent security concerns. We focused on the honest-but-curious cloud model and we proposed MapReduce protocols that enjoy the security guarantee that none of the cloud nodes can learn the input or the output data. The goal of this chapter is to provide an overview of our protocols, which span fundamental problems such as matrix multiplication and relational query evaluation (joins, grouping and aggregation, set intersection). To illustrate the challenges of our problem setting and our contributions, in this chapter we mainly rely on matrix multiplication, and we give a glimpse on the other protocols.

Relevant Publications

- Research papers: ARES [BCGL17], SECRYPT/ICETE [CGLY18, CGLY19c, CGLY19b, CGLY19a], FPS [BCG+18]

Contents



(a) Cloud computations with MapReduce.

Query	*MapReduce Input*	*MapReduce Output*
Matrix multiplication	Matrices M, N	Matrix $P = M \cdot N$
Natural join	Relations $R_1, \ldots, R_n$	Relation $R_1 \bowtie \ldots \bowtie R_n$
Group by attribute A and apply aggregate function θ on attribute B	Relation R	Relation $\gamma_{A,\theta(B)}(R)$
Set intersection	Relations $R_1, \ldots, R_n$	Relation $R_1 \cap \ldots \cap R_n$

(b) MapReduce problems for which we proposed secure protocols.

Figure 5.1: Outsourcing MapReduce data and computations to the cloud.

5.1 Context

MapReduce [DG04] is a programming paradigm that allows big data processing. MapReduce users need to specify two functions (*map* and *reduce*) that are executed in parallel on a large cluster of commodity machines. The MapReduce environment takes care of technical aspects such as partitioning the input data, scheduling the program's execution across the machines, handling machine failures, and managing the network communication. The popularity of the MapReduce paradigm is due to the fact that the users do not need to handle such technical aspects that usually make distributed programming hard.

MapReduce users often outsource their data and computations to a public cloud, which makes big data processing accessible to users who cannot afford building their own clusters, but yields inherent security and privacy issues because users are no longer in control of their data. Hence, the outsourced data may be communicated over an untrustworthy network and may be stored on some cloud machines where curious cloud admins may have access and learn some sensitive data. Investigating the security and privacy aspects of MapReduce computations in public clouds is a hot research topic, see [DDGS16] for a recent survey on different adversarial models and state-of-the-art systems. As noted in [DDGS16]: "*MapReduce was designed to be deployed on-premises under mistaken assumption that local environment can be completely trusted. Thus, security and privacy aspects were overlooked in the initial design. As MapReduce gained popularity the lack of security and privacy in on-premises deployment become severe shortcoming.*"

We depict in Figure 5.1(a) the scenario that we consider for cloud computation with MapReduce. Our scenario is coherent with those considered in Chapter 3 and 4, on securing SPARQL query evaluation and bandit problems, respectively. Our scenario is also inspired by state-of-the-art systems for private and secure MapReduce e.g., [DLS16].

More precisely, we assume two types of users:

- *Data owner*, who outsources some data (i.e., the MapReduce input) to the cloud.
- *Data client*, who is allowed to submit some query over the data owner's data and receives the query result (i.e., the MapReduce output) from the cloud.

The data storage and computations are hence outsourced to the cloud, that we assume *honest-but-curious* i.e., it performs the assigned computations correctly, without modifying or deleting the data, but tries to learn as much as possible from the data that it sees. This type of adversary

is a classical model for public cloud providers, which is relevant for both MapReduce security and privacy aspects surveyed in [DDGS16], as outlined in Section 1.

The goal of this chapter is to report on our research on developing MapReduce protocols that are executed on an honest-but-curious cloud while enjoying the desirable security (and privacy) property that the cloud nodes cannot learn neither the MapReduce input or output. Our approach was to study standard MapReduce algorithms [LRU14, Chapter 2], understand what are the needed computations, and look for practical cryptographic schemes that have desirable properties depending on the needed computations. Then, we put all pieces together in order to propose secure protocols that should give exactly the same output as the standard MapReduce algorithms, while hiding the input and output from the honest-but-curious cloud.

5.2 Related Work

[LRU14, Chapter 2] presents an introduction to the MapReduce paradigm, as well as simple (non-secure) MapReduce algorithms that solve some standard problems. The security and privacy concerns of MapReduce have been recently surveyed [DDGS16]. The general goal of state-of-the art techniques e.g., [BDPMÖ12, DLS16, MBC13, TNP15, VBN19] is to execute MapReduce computations such that the public cloud cannot learn the data. This is our goal too, but for different problems: matrix multiplication, natural joins on arbitrary number of relations, grouping and aggregation, and set intersection. We summarize in Figure 5.1(b) the problems for which we proposed secure protocols.

Regarding MapReduce *matrix multiplication*, note that it generalizes matrix-vector multiplication i.e., the original purpose for which Google created MapReduce because such multiplications are needed for PageRank computation. In [BCGL17], we enhanced with security guarantees two standard MapReduce matrix multiplication algorithms given in [LRU14, Chapter 2] (with two rounds and one round, respectively). Then, in [CGLY19c, CGLY19a], we proposed a secure MapReduce protocol for Strassen-Winograd matrix multiplication [vG13, Chapter 12], which has sub-cubic complexity, hence is more efficient than the standard naive matrix multiplication algorithm that has cubic complexity. To the best of our knowledge, no existing work has addressed the problem of secure matrix multiplication with MapReduce. From a different point of view, distributed matrix multiplication has been investigated in the secure multi-party computation model, whose goal is to allow different nodes to jointly compute a function over their private inputs without revealing them. Existing works on secure distributed matrix multiplication e.g., [DLOP17] have different assumptions compared to our MapReduce framework: (i) they assume that nodes contain entire vectors, whereas the division of the initial matrices in chunks as done in MapReduce does not have such assumptions, and (ii) in MapReduce, the functions specified by the user [DG04] are limited to *map* (process a key/value pair to generate a set of intermediate key/value pairs) and *reduce* (merge all intermediate values associated with the same intermediate key), and the matrix multiplication is done in two or one MapReduce rounds [LRU14]; on the other hand, the works in the multi-party computation model assume arbitrary numbers of rounds, relying on more complex functions than map and reduce.

Furthermore, regarding *relational query evaluation*, we proposed secure MapReduce protocols for natural joins [BCG+18], grouping and aggregation [CGLY18], and set intersection [CGLY19b]. In Figure 5.1(b), by $\bowtie$ we denote the natural join, by γ we denote the grouping and aggregation, and θ can be any standard SQL aggregation function (count, avg, sum, min, max). For all aforementioned problems, standard MapReduce algorithms are given in [LRU14, Chapter 2]. For joins, we secured two standard n-ary generalizations of the binary MapReduce algorithm: cascade of $n-1$ binary joins vs all joins at once with a hypercube approach [CBS15].

Concerning our work on secure joins with MapReduce [BCG+18], a closely related work is CryptDB [PRZB11]. Indeed, we also rely on the ideas of encrypting the join attributes with a deterministic encryption scheme in order to perform equijoins, and moreover, encrypting the other attributes with an IND-CPA encryption scheme. On the other hand, CryptDB does not consider the MapReduce paradigm. Another related work is [DLS16], which proposes a secure protocol for performing binary joins with MapReduce. The assumptions of [DLS16] are different than ours because it does computations on secret shares [Sha79] in the cloud and the data client performs the interpolation on the outputs. On the other hand, we aim at outsourcing the bulk of the computations to the cloud and minimizing the computations done by the data client.

Moreover, regarding our work on secure grouping and aggregation with MapReduce [CGLY18], a closely related work is also CryptDB [PRZB11] as we also rely on the ideas of using deterministic encryption for the group by attributes, order-preserving encryption for min/max aggregates, and additive homomorphic encryption for the other aggregates. However, as already mentioned, CryptDB does not consider the MapReduce paradigm. From a different point of view, secure and distributed grouping and aggregation has been also studied outside the MapReduce paradigm using trusted hardware [TNP16].

Finally, regarding our work on secure set intersection with MapReduce [CGLY19b], CryptDB is no longer a related work as it does not consider set intersection. From a different point of view, there are connections between our work and private set intersection [FNP04], where two parties compute the intersection of their respective sets while revealing minimal information about their sets. To the best of our knowledge, there is no existing work on private set intersection with MapReduce. An important different assumption between the MapReduce setting that we consider and the private set intersection line of work is that we aim for a MapReduce protocol that does not reveal the intersection result to the data owners; only the data client is allowed to query the intersection result whose computation is outsourced to the cloud.

5.3 Summary of Contributions

We give an overview of our contributions on matrix multiplication with two MapReduce rounds in Section 5.3.1 and of our other contributions in Section 5.3.2.

5.3.1 Matrix Multiplication with Two MapReduce Rounds

Matrix multiplication. Let $M_{a,b}$ and $N_{b,c}$ be two compatible matrices, and let $P_{a,c}$ be the result of the matrix multiplication $M \cdot N$. We denote by m_{ij} the element of M on the i^{th} line and j^{th} column, and we use similar notations for denoting the elements of N and P. In particular, $p_{ik} = \sum_{j=1}^{b} m_{ij} \cdot n_{jk}$ (for $1 \leq i \leq a$ and $1 \leq k \leq c$). For example:

$$\begin{array}{ccccc} M_{2,2} & \cdot & N_{2,3} & = & P_{2,3} \\ \begin{bmatrix} 1 & 2 \\ 0 & 5 \end{bmatrix} & \cdot & \begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & 6 \end{bmatrix} & = & \begin{bmatrix} 2 & 2 & 15 \\ 5 & 0 & 30 \end{bmatrix} \end{array}$$

Matrix multiplication with MapReduce. Standard MapReduce matrix multiplication algorithms are given in [LRU14, Chapter 2]. The simplest algorithm uses two rounds. A *MapReduce round* consists of a sequence of applying a *map* function in parallel to a set of inputs, followed by applying a *reduce* function in parallel to the outputs of the map. Intuitively, the map function outputs a set of key-value pairs, whereas the reduce function takes as input a key and the collection of all values associated to it (coming from all applications of map) and produces a new

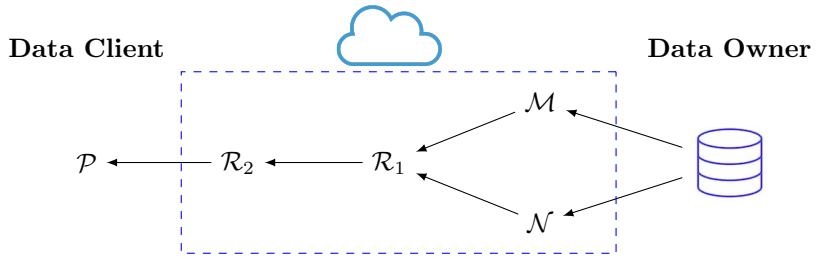


Figure 5.2: Matrix multiplication with 2 MapReduce rounds.

value that aggregates the collection of input values. We summarize in Figure 5.2 the workflow for computing matrix multiplication with 2 MapReduce rounds, that we detail next. Moreover, in Figure 5.3 we show a detailed example of the data flow from the input until the output, including all key-value pairs communicated between map and reduce nodes.

For storing matrices in the distributed file system of the MapReduce environment, we assume a standard data structure consisting of a ternary relation, where the first two columns encode the position (line and column) of each element, whereas the last column stores the actual elements. For example, the aforementioned input M and N, and output P are stored as shown in Figure 5.3. We assume that the data owner outsources matrices M and N to the public cloud, which stores them on two sets of nodes $\mathcal{M}$ and $\mathcal{N}$, respectively. Moreover, we assume that the output matrix P is stored on a set of data client's nodes $\mathcal{P}$. Next, we explain what each MapReduce round does (see [BCGL17] for pseudocode). Intuitively, Round 1 computes on some cloud nodes $\mathcal{R}_1$ all needed products of elements from M and N, whereas Round 2 computes on some cloud nodes $\mathcal{R}_2$ the elements of P by summing the products output of Round 1. More formally, we have:

- **Round 1**

  Map: $\mathcal{M}$ emits to $\mathcal{R}_1$: $\{(j, (\mathtt{M}, i, m_{ij}))\}_{1\leq i\leq a, 1\leq j\leq b}$

  $\mathcal{N}$ emits to $\mathcal{R}_1$: $\{(j, (\mathtt{N}, k, n_{jk}))\}_{1\leq j\leq b, 1\leq k\leq c}$

  ($\mathtt{M}$ and $\mathtt{N}$ are bits indicating the provenance matrix of each emitted element)

  Reduce: $\mathcal{R}_1$ emits to $\mathcal{R}_2$: $\{((i,k), m_{ij}\cdot n_{jk})\}_{1\leq i\leq a, 1\leq j\leq b, 1\leq k\leq c}$

- **Round 2**

  Map: this is simply the identity function, which copies the received input on the output.

  Reduce: $\mathcal{R}_2$ emits to $\mathcal{P}$: $\{(i,k), p_{ik} = \sum_{j=1}^{b} m_{ij}\cdot n_{jk}\}_{1\leq i\leq a, 1\leq k\leq c}$

Security aspects. We instantiate the generic setting outlined in Section 5.1 to the problem of matrix multiplication with 2 MapReduce rounds. Hence, we assume an honest-but-curious cloud and we aim for the following security property: cloud nodes $\mathcal{M}$, $\mathcal{N}$, $\mathcal{R}_1$, and $\mathcal{R}_2$ cannot learn matrices M, N, and P. A natural way to approach this problem setting is to rely on a *fully-homomorphic encryption* i.e., to encrypt the data and then perform computations directly on ciphertexts. In particular, the computations needed in our setting are multiplications ($\cdot$) in the reduce of Round 1 and additions ($+$) in the reduce of Round 2. Ideally, we would like that before outsourcing matrices M and N to the public cloud, the data owner encrypts all their

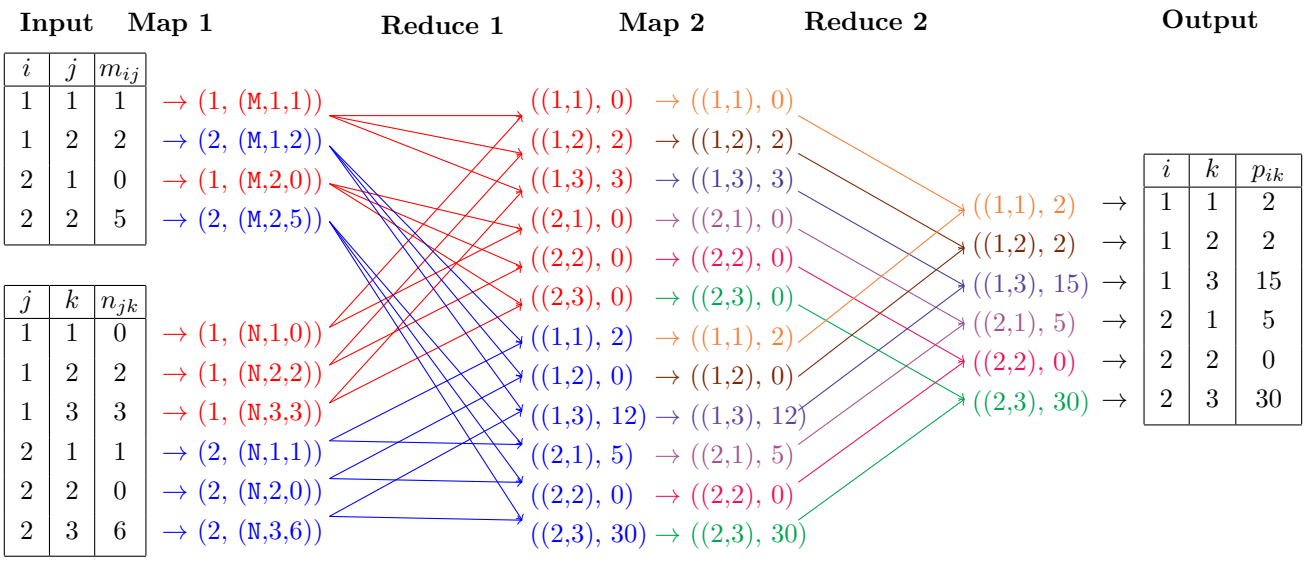


Figure 5.3: Example of data flow for matrix multiplication with 2 MapReduce rounds.

values using a fully-homomorphic encryption [Gen09] i.e., a function $\mathcal{E}$ such that $\mathcal{E}(m_1 + m_2)$ and $\mathcal{E}(m_1 \cdot m_2)$ can be computed directly from $\mathcal{E}(m_1)$ and $\mathcal{E}(m_2)$. Such an approach would solve our problem only from a theoretical point of view. Indeed, as mentioned in the previous chapter, it remains an open question how to make such a scheme work fast and be accurate in practice, and the research and development community [SEA, HEl] is actively working on this topic.

Consequently, our approach is to rely on *partially-homomorphic encryption* and extend it to fill our needs for adding security guarantees to MapReduce matrix multiplication algorithms. We rely on *Paillier's cryptosystem* [Pai99], also used in Section 4.3.1. We recall that Paillier is asymmetric i.e., encryption and decryption are done using two different keys: public key (pk) and secret key (sk), respectively. Moreover, Paillier is additive homomorphic i.e., $\mathcal{E}_{\mathsf{pk}}(m_1+m_2) = \mathcal{E}_{\mathsf{pk}}(m_1) \cdot \mathcal{E}_{\mathsf{pk}}(m_2)$. However, to obtain the ciphertext of a multiplication, one of the two operands is needed in plain format to be able to compute an exponentiation: $\mathcal{E}_{\mathsf{pk}}(m_1 \cdot m_2) = (\mathcal{E}_{\mathsf{pk}}(m_1))^{m_2}$. Hence, it is not difficult to see that a naive use of Paillier for MapReduce matrix multiplication would reveal one of matrices M or N to the cloud. Furthermore, Paillier is IND-CPA secure and all theoretical security properties of our protocols also hold if we choose any other IND-CPA asymmetric, additive homomorphic scheme instead of Paillier; our choice is due to practical reasons seen that Paillier is implemented in a number of libraries that can be used in practice.

Secure-Private (SP) protocol for matrix multiplication with 2 MapReduce rounds. Our SP protocol relies on Paillier and satisfies the desired security property. A first important step is the outsourcing of matrices M and N by the data owner in encrypted format, using two different techniques, as we explain next. We assume that the data client generates a key pair $(\mathsf{pk}_{\mathcal{P}}, \mathsf{sk}_{\mathcal{P}})$ and the cloud nodes of type $\mathcal{R}_2$ generate a key pair $(\mathsf{pk}_{\mathcal{R}_2}, \mathsf{sk}_{\mathcal{R}_2})$. We depict the general workflow in Figure 5.4, where we annotate each of the participants with the keys that they know. In particular, the secret keys $\mathsf{sk}_{\mathcal{P}}$ and $\mathsf{sk}_{\mathcal{R}_2}$ are obviously known only by the participants that generate them (the data client and $\mathcal{R}_2$, respectively), whereas the public keys

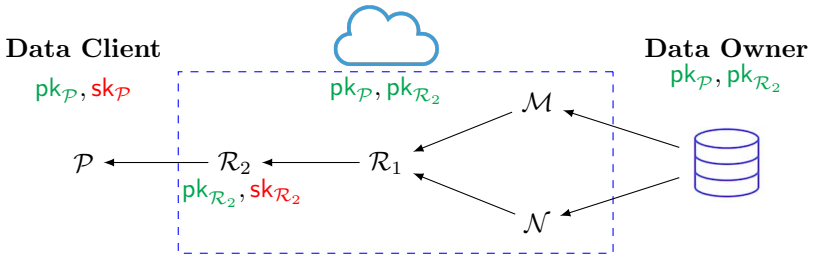


Figure 5.4: SP matrix multiplication with 2 MapReduce rounds.

$\mathsf{pk}_{\mathcal{P}}$ and $\mathsf{pk}_{\mathcal{R}_2}$ are publicly available to the data owner and to all cloud nodes. We next revisit the map and reduce functions given earlier to present their SP versions (differences are highlighted):

- **Round 1**

  Map: $\mathcal{M}$ emits to $\mathcal{R}_1$: $\{(j, (\mathtt{M}, i, \mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(m_{ij})))\}_{1 \le i \le a, 1 \le j \le b}$

  $\mathcal{N}$ emits to $\mathcal{R}_1$: $\{(j, (\mathtt{N}, k, n_{jk}+\tau_{jk}, \mathcal{E}_{\mathsf{pk}_{\mathcal{R}_2}}(\tau_{jk})))\}_{1 \le j \le b, 1 \le k \le c}$

  Recall that the encryption has been done by the data owner before outsourcing and the map only rewrites the encrypted matrices in a suitable key-value format.

  Reduce: $\mathcal{R}_1$ emits to $\mathcal{R}_2$:

  $\{((i,k), ((\mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(m_{ij}))^{n_{jk}+\tau_{jk}}, \mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(m_{ij}), \mathcal{E}_{\mathsf{pk}_{\mathcal{R}_2}}(\tau_{jk})))\}_{1 \le i \le a, 1 \le j \le b, 1 \le k \le c}$

- **Round 2**

  Map: this is again the identity function.

  Reduce: $\mathcal{R}_2$ computes

$$\prod_{j=1}^{b} \frac{(\mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(m_{ij}))^{n_{jk}+\tau_{jk}}}{(\mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(m_{ij}))^{\tau_{jk}}} = \prod_{j=1}^{b} \mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(m_{ij} \cdot n_{jk}) = \mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(p_{ik}).$$

  These equalities follow from the additive homomorphic property of Paillier. Moreover, since $\mathcal{R}_2$ has $\mathsf{sk}_{\mathcal{R}_2}$, it can compute the mask τ_{jk} as $\mathcal{D}_{\mathsf{sk}_{\mathcal{R}_2}}(\mathcal{E}_{\mathsf{pk}_{\mathcal{R}_2}}(\tau_{jk}))$. Then, $\mathcal{R}_2$ emits to $\mathcal{P}$: $\{(i,k), \mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(p_{ik})\}_{1 \le i \le a, 1 \le k \le c}$. Finally, the data client retrieves P by decrypting each $\mathcal{E}_{\mathsf{pk}_{\mathcal{P}}}(p_{ik})$ using $\mathsf{sk}_{\mathcal{P}}$.

Contrarily to the standard MapReduce matrix multiplication that reveals to the cloud the input and output matrices, the SP protocol satisfies the desired security property. We next intuitively explain why cloud nodes $\mathcal{M}$, $\mathcal{N}$, $\mathcal{R}_1$, and $\mathcal{R}_2$ cannot learn matrices M, N, and P (provided that the cloud nodes do not collude), and we prove it formally in [BCGL17].

- Cloud nodes $\mathcal{M}$ see each element of the matrix M encrypted with the data client's public key $\mathsf{pk}_{\mathcal{P}}$. Since $\mathcal{M}$ does not have $\mathsf{sk}_{\mathcal{P}}$, it cannot learn M.
- Cloud nodes $\mathcal{N}$ see each element of the matrix N with a random mask, as well as the encryption of the mask using the public key $\mathsf{pk}_{\mathcal{R}_2}$ of the cloud nodes $\mathcal{R}_2$. Since $\mathcal{N}$ does not have $\mathsf{sk}_{\mathcal{R}_2}$, it cannot learn N.
- Cloud nodes $\mathcal{R}_1$ see each element of M encrypted with $\mathsf{pk}_{\mathcal{P}}$, each element of N with a random mask, and the encryption of the mask using the $\mathsf{pk}_{\mathcal{R}_2}$. Since $\mathcal{R}_1$ has neither $\mathsf{sk}_{\mathcal{P}}$ nor $\mathsf{sk}_{\mathcal{R}_2}$, it can learn neither M nor N.

Alice has $(\mathsf{sk}, \mathsf{pk})$		Bob has $c_1 = \mathcal{E}_{\mathsf{pk}}(m_1)$ and $c_2 = \mathcal{E}_{\mathsf{pk}}(m_2)$ and needs to compute $\mathcal{E}_{\mathsf{pk}}(m_1 \cdot m_2)$
		Pick two randoms δ_1, δ_2
$\mathcal{D}_{\mathsf{sk}}(\alpha_1) = m_1 + \delta_1$	$\xleftarrow{\alpha_1, \alpha_2}$	$\alpha_1 = c_1 \cdot \mathcal{E}_{\mathsf{pk}}(\delta_1)$
$\mathcal{D}_{\mathsf{sk}}(\alpha_2) = m_2 + \delta_2$		$\alpha_2 = c_2 \cdot \mathcal{E}_{\mathsf{pk}}(\delta_2)$
$\beta = \mathcal{E}_{\mathsf{pk}}((m_1 + \delta_1) \cdot (m_2 + \delta_2))$	$\xrightarrow{\beta}$	$\frac{\beta}{\mathcal{E}_{\mathsf{pk}}(\delta_1 \cdot \delta_2) \cdot c_1^{\delta_2} \cdot c_2^{\delta_1}} = \mathcal{E}_{\mathsf{pk}}(m_1 \cdot m_2)$

Figure 5.5: Interactive homomorphic multiplication of ciphertexts [CDN01] such that neither Alice nor Bob learns m_1 or m_2.

- Cloud nodes $\mathcal{R}_2$ have $\mathsf{sk}_{\mathcal{R}_2}$ thus can decrypt the mask, but cannot learn elements of N without colluding with nodes $\mathcal{N}$. This is true because nodes $\mathcal{R}_2$ do not see masked elements of N, but only see results of exponentiation using masked elements of N. Moreover, since $\mathcal{R}_2$ does not have $\mathsf{sk}_{\mathcal{P}}$, it cannot learn output P.

Additionally, we observe that the data client cannot learn matrices M and N because the data client's nodes $\mathcal{P}$ receive only the output matrix P encrypted with $\mathsf{pk}_{\mathcal{P}}$ hence the data client never sees the input matrices, which have been encrypted by the data owner before outsourcing.

Impact of collusions. The SP protocol satisfies the desired security property that none of the cloud nodes can learn the input or output matrices. This property holds under the hypothesis that the cloud nodes do not collude. In particular, if cloud nodes $\mathcal{N}$ and $\mathcal{R}_2$ collude, they can learn the input matrix N because nodes $\mathcal{R}_2$ can decrypt the masks applied to elements of matrix N, and consequently nodes $\mathcal{N}$ can remove the masks from each element.

Since we assume the *honest-but-curious* cloud model, we may choose to not consider collusions between nodes as we did in the previous two chapters, where we cited a classical formulation [Gol04, Chapter 7] (where honest-but-curious is denoted semi-honest). However, we believe that it is important to consider collusions for our secure MapReduce protocols, as we argue next. On the one hand, the secure protocols from the previous two chapters (on securing SPARQL query evaluation and bandit problems, respectively) have in common the approach of proposing ad hoc distribution strategies, which makes easier to enforce the no-collusion hypothesis. On the other hand, in this chapter, the distribution strategy is not ad hoc, but is dictated by the MapReduce paradigm. This means that a same cloud node that is used to store a chunk of the input matrices may be also used as a computing node for map and/or reduce functions, at Round 1 and/or at Round 2. Hence, it may happen that a same cloud node sees the data manipulated by $\mathcal{N}$ and $\mathcal{R}_2$, which implies a collusion that leaks elements of matrix N. Consequently, we believe that it is important to consider collusions in our cloud model for secure MapReduce, which means that we need to refine our SP protocol to make it *collusion-resistant*.

Collusion-Resistant-Secure-Private (CRSP) protocol for matrix multiplication with 2 MapReduce rounds. The CRSP protocol satisfies the desired security property (none of the cloud nodes can learn the input or output matrices) even in the presence of collusions between cloud nodes. The key ideas of CRSP are: (i) before matrix outsourcing, the data owner encrypts both matrices M and N using the public key $\mathsf{pk}_{\mathcal{P}}$ of the data client, and (ii) the multiplications from the reduce of Round 1 are done using a well-known protocol for interactive homomorphic multiplication of ciphertexts [CDN01], outlined in Figure 5.5. We next revisit the map and reduce functions given earlier to present their CRSP versions (differences are highlighted):

- **Round 1**

  Map: $\mathcal{M}$ emits to $\mathcal{R}_1$: $\{(j, (\mathtt{M}, i, \mathcal{E}_{\mathsf{pk}_\mathcal{P}}(m_{ij})))\}_{1 \leq i \leq a, 1 \leq j \leq b}$

  $\mathcal{N}$ emits to $\mathcal{R}_1$: $\{(j, (\mathtt{N}, k, \mathcal{E}_{\mathsf{pk}_\mathcal{P}}(n_{jk})))\}_{1 \leq j \leq b, 1 \leq k \leq c}$

  Reduce: $\mathcal{R}_1$ uses *interactive homomorphic multiplication* cf. Figure 5.5 and emits to $\mathcal{R}_2$:

  $\{((i, k), \mathcal{E}_{\mathsf{pk}_\mathcal{P}}(m_{ij} \cdot n_{jk}))\}_{1 \leq i \leq a, 1 \leq j \leq b, 1 \leq k \leq c}$

- **Round 2**

  Map: this is again the identity function.

  Reduce: $\mathcal{R}_2$ emits to $\mathcal{P}$: $\{(i, k), \mathcal{E}_{\mathsf{pk}_\mathcal{P}}(p_{ik}) = \prod_{j=1}^{b} \mathcal{E}_{\mathsf{pk}_\mathcal{P}}(m_{ij} \cdot n_{jk})\}_{1 \leq i \leq a, 1 \leq k \leq c}$

In [BCGL17], we analyzed the computation and communication costs for SP and CRSP protocols: for each complexity measure and each protocol, the overhead with respect to the standard (non-secure) version is a constant factor. The obvious trade-off is that CRSP guarantees stronger security properties (i.e., it resists to collusions), at the price of an increased complexity, and also at the price of interactions with the data client that needs to do more computations than just decrypting the final result. Whereas [BCGL17] is a theoretical paper presenting pseudocode and analysis of our protocols, a proof-of-concept empirical evaluation presented in [Gir19] confirms the theoretical analysis and suggests the feasibility of our protocols.

5.3.2 Other Protocols

Other contributions on matrix multiplication. In addition to the two-rounds MapReduce matrix multiplication algorithm for which we described secure protocols in Section 5.3.1, [LRU14, Chapter 2] also proposes an one-round algorithm, known to be typically less efficient than the two-rounds algorithm. The trade-off is that the one-round algorithm has less rounds at the price of an increased complexity because the map and reduce functions from the unique round are heavier; in particular, redundant data is emitted by the map. In [BCGL17], we also proposed secure protocols for the one-round algorithm. Since the multiplications and additions are done in the same round on the same reduce node, the technique used for the SP protocol in Section 5.3.1 is not anymore useful because the matrix N would be leaked to the cloud. On the other hand, the same technique that we used for the CRSP protocol in Section 5.3.1 is still useful to propose a CRSP protocol for one-round MapReduce matrix multiplication, which satisfies the desired security property even in the presence of collusions [BCGL17].

The two MapReduce matrix multiplication algorithms presented in [LRU14, Chapter 2] are MapReduce variants of the standard naive matrix multiplication algorithm that has cubic complexity. Seen that many research efforts have been made to propose more efficient matrix multiplication [vG13, Chapter 12], we investigated whether we can propose a secure MapReduce protocol for some sub-cubic algorithm. We relied on the well-known Winograd variant of the Strassen algorithm [vG13, Chapter 12], which has a complexity of $O(n^{\log_2^7})$ for matrices of size $n \times n$. The complexity decrease is achieved by handling multiplications recursively and saving one (costly) multiplication at the expense of some (cheap) additions. We proposed a MapReduce version of the Strassen-Winograd algorithm, that we subsequently secured using the same technique that we used for the CRSP protocol in Section 5.3.1. Hence, we improved the efficiency of CRSP protocol from [BCGL17] while keeping the same security guarantees [CGLY19c, CGLY19a].

Natural joins. As briefly mentioned in Section 5.2, we added security guarantees to two standard n-ary generalizations of the binary MapReduce join algorithm [LRU14, Chapter 2]:

- Cascade: a sequence of binary joins; to join n relations, we need $n-1$ MapReduce rounds.
- Hypercube [CBS15]: evaluating all joins using a single MapReduce round.

The trade-off between the two approaches is that Hypercube uses less rounds at the price of redundant data emitted by the unique map, whereas Cascade does not emit such redundant data but may manipulate large intermediate results containing tuples that are not used in the final join result. None of the two approaches consistently wins over the other in practice because the actual performance depends on the data and query [CBS15].

We refer to [BCG+18] for our secure protocols, as well as their theoretical analysis and a proof-of-concept empirical evaluation. We next give some intuition about our contributions.

In our first protocol, we require the data owner to outsource the relations in encrypted format, by relying on two techniques: (i) deterministic encryption with AES [AES01] of the join attributes, and (ii) non-deterministic IND-CPA encryption of all attributes, using the public key of the data client; in our implementation we relied on RSA-OAEP [BR94]. The use of deterministic encryption for the join attributes allows the cloud to perform equality tests hence to run the join algorithms (Cascade or Hypercube) directly in the encrypted domain. At the end, the cloud sends to the data client the join result encrypted with the data client's public key, hence the data client can decrypt and see the join result. Our protocol has the desirable security property that none of the cloud nodes can learn the input relations or the output relation, even in the presence of collusions between the cloud nodes. However, if the data client colludes with the cloud, then they can learn input tuples that are not present in the output join result, which can be considered as a security breach of the data owner.

Hence, we proposed a second protocol to counter such an additional attack, where we introduced a trusted proxy, which colludes neither with the cloud nor with the data client. Before data outsourcing, we require the data owner to do all the encryption from the first protocol, and additionally encrypt the data from the aforementioned (ii) using the public key of the proxy. Then, the cloud does the join computation as for the first protocol. At the end, the cloud sends the result to the proxy, who uses its secret key to decrypt and then sends the result to the data client encrypted only with the public key of the data client.

Grouping and aggregation. In [CGLY18], we discuss how to add security guarantees to the standard grouping and aggregation MapReduce algorithms [LRU14, Chapter 2]. We next give some intuition about our protocols. Before outsourcing, the data owner encrypts the group by attribute using deterministic encryption, to ensure that the map emits to the same reducer all tuples sharing the same value of the group by attribute. Moreover, the data owner also encrypts the group by attribute using some public key IND-CPA encryption using the public key of the data client. Hence, the data client can decrypt the group by attribute from the query result. Also before outsourcing, the data owner encrypts attributes used for min/max aggregates using order-preserving encryption. Then, the cloud runs the standard MapReduce grouping and aggregation algorithms on the pre-processed outsourced data, with the only change that for count/sum/avg, the map emits values encrypted with some public key IND-CPA additive homomorphic encryption using the public key of the data client. Hence, the reduce can compute count and sum directly in the encrypted domain, which are emitted encrypted to the data client, who can decrypt and see the results; for avg, the data client receives both sum and count, and computes their division on her side. By relying on the aforementioned techniques, we obtain protocols with the security property that the cloud nodes cannot learn the input or output, which also holds in the presence of collusions between cloud nodes. In practice, one may choose no matter what encryption schemes that satisfy the needs of our protocols e.g., AES [AES01]

for deterministic encryption, Paillier [Pai99] for additive homomorphic public key IND-CPA encryption, and [BCLO09] for order-preserving encryption.

Set intersection. In [CGLY19b], we assume n data owners such that each of them does not want to share its set with the others, and a data client that is allowed to query the intersection of the n sets. The computation of the set intersection is outsourced to the cloud. We propose a secure MapReduce protocol that computes the intersection of the n sets such that the cloud does not learn any input or output data. The property also holds in the presence of collusions between cloud nodes; if the cloud and the data client collude, they cannot learn input elements that do not appear in the intersection. Our technique relies on a pre-processing step that each data owner does before outsourcing their respective relations, using the following cryptographic techniques. (i) A randomly chosen data owner encrypts each element of its set with the public key of the data client; then, the result is xor-ed with the results of $n-1$ AES encryptions of the element, using $n-1$ distinct keys previously shared with the other data owners. (ii) Each of the other $n-1$ data owners encrypts each element of its set using a single AES key among the aforementioned $n-1$ keys. Intuitively, our technique works because if an element has to be part of the intersection, then it should be outsourced by all n data owners; hence, by xor-ing the n encrypted values received from the n data owners, the cloud gets precisely the input element encrypted with the public key of the data client; then, the data client can decrypt the elements that it receives at the end of the protocol as part of the intersection. We refer to [CGLY19b] for more details on the analysis our protocol, as well as a proof-of-concept empirical evaluation.

5.4 Conclusions

We presented an overview of our work on secure MapReduce protocols, a research direction on which my main collaborators were Pascal Lafourcade (LIMOS) and Matthieu Giraud, a PhD student that we co-advised. The other collaborators were Xavier Bultel and Lihua Ye, two other students (PhD and MSc, respectively) of Pascal. Our work lead to research papers at ARES [BCGL17], SECRYPT/ICETE [CGLY18, CGLY19c, CGLY19b, CGLY19a], and FPS [BCG+18]. My contributions were mainly conceptual, as I participated to defining the problem settings, designing the protocols, and analyzing their complexity.

As future work, we plan to investigate whether our techniques could be generalized to work on other big data paradigms such as Spark [KKWZ15]. Indeed, Spark is an open source distributed general-purpose cluster computing framework that became increasingly more popular than MapReduce because it is faster and easier to use. The same security and privacy concerns surveyed in [DDGS16] for MapReduce also seem to hold for Spark, although the security and privacy of Spark have not been addressed yet by many works, except e.g., [SPZ16]. Hence, it would be interesting to see whether our works on secure MapReduce protocols for matrix multiplication, joins, grouping and aggregation, and set intersection could be done more efficiently on Spark and also generalized to secure other types of problems.

Chapter 6

Conclusions and Perspectives

Conclusions

To sum up my research, I had a first research cycle (2012–16) in the data management community, followed by a second cycle (since 2016) when I additionally explored the security community and I also worked with new machine learning tools such as multi-armed bandits. The goal of this document was to give an overview of my second research cycle:

Chapter 2. I summarized the design and implementation principles of the gMark and EGG systems. gMark [BBC+17a, BBC+17b, BBC+16] is a domain- and query language-independent synthetic generator of graphs and query workloads. gMark is the first generator that supports unions of conjunctions of regular path queries (a fundamental graph query paradigm including recursive queries), and moreover, supports schema-driven query selectivity estimation. Furthermore, EGG [ACM17a, ACM17b] relies on gMark as a building block in order to generate evolving graphs based on finely tuned temporal constraints.

Chapter 3. I presented an overview of the workflow, theoretical analysis, and empirical evaluation of GOOSE [CL20b, CL20a], a secure framework for graph outsourcing and SPARQL evaluation. GOOSE relies on cryptographic schemes and secure multi-party computation to achieve desirable security properties: (i) no cloud node can learn the graph, (ii) no cloud node can learn at the same time the query and the query answers, and (iii) an external network observer cannot learn the graph, the query, or the query answers. The large-scale empirical evaluation of GOOSE benefited from the existence of gMark (cf. Chapter 2).

Chapter 4. I summarized my research on secure and distributed protocols for multi-armed bandits, a popular class of sequential machine learning algorithms. I mainly focused on the UCB-DS protocol for secure cumulative reward maximization for standard stochastic bandits [CLLS20]. I also included a brief overview of our protocols for cumulative reward maximization for linear bandits [CDLS20] and best arm identification [CLLS19]. Each of our protocols returns the same output as the standard, non-secure algorithms, while guaranteeing that the cloud cannot learn more than limited pieces of the input and output.

Chapter 5. I gave an overview of my work on secure protocols for MapReduce, a popular programming paradigm for big data processing. To illustrate the challenges and the contributions, I mainly focused on matrix multiplication [BCGL17, CGLY19c, CGLY19a], and then I also presented a glimpse on our protocols for relational query evaluation tasks: natural joins [BCG+18], grouping and aggregation [CGLY18], and set intersection [CGLY19b].

Our protocols return the same result as the standard, non-secure algorithms, while guaranteeing that none of the cloud nodes can learn the MapReduce input and output.

The aforementioned works yielded to publications relevant for several communities: *data management* (TKDE journal, VLDB demo, ICDE poster), *semantic Web* (two ISWC demos), and *security* (research papers in TrustCom, DBSec, ProvSec, ISPEC, SECRYPT, ARES, FPS). Hence, my second research cycle had a major thematic shift with respect to the first one, which was purely focused on the data management community (TODS journal, research and demo papers in EDBT, VLDB and SIGMOD). I included my list of publications in Section 7.1.

Perspectives

During the next few years, I plan to pursue research relevant to the data management community, while exploiting what I learned on security and machine learning, and while exploring new topics. Next, I briefly discuss three research directions that I envision to tackle; each of them could potentially involve the supervision of a PhD thesis, depending on the availability of good candidates and of funding. The enumeration of research directions is not exhaustive seen that I may start new collaborations on new topics.

Secure outsourced graph query evaluation. The goal of this research direction (related to Chapters 2, 3, and 5) is to design and implement a system capable of securely and efficiently evaluating graph queries on outsourced graphs. As discussed in Chapter 3, it would be interesting to extend GOOSE to support queries more complex than UCRPQ, and moreover, by analyzing the performance of GOOSE, we observed that the overhead due to cryptographic primitives is dominated by the time taken by the graph query engine used as a black-box in GOOSE. From a different point of view, in Chapter 5, we mentioned that it would be interesting to propose secure protocols for outsourced Spark programs for query evaluation, in the spirit of our works on secure MapReduce. By exploring a synergy between the two topics, a natural idea is to propose a system that evaluates graph queries more complex than UCRPQ using Spark and while protecting the data security. This research direction could lead to theoretical contributions (protocols with formally-proven properties in terms of complexity and security) and also practical contributions (system prototype available open source for the community). Moreover, in order to realize a principled large-scale empirical evaluation of the new system, it is important to extend gMark (cf. Chapter 2) to take into account more complex queries.

Secure sequential and federated learning. This research direction is related to Chapter 4 and also implies the exploration of new topics such as federated learning. First, we plan to continue our work on secure protocols for multi-armed bandits (cf. Chapter 4) by securing bandit algorithms for bandits with a more complex structure [LS20]. Then, we plan to work on securing algorithms relevant to other types of sequential learning, such as reinforcement learning [SB98]. From a different point of view, we are interested in contributing to other models of outsourced machine learning than the cloud that we are currently investigating. More precisely, we think at proposing secure protocols for *federated learning*, a novel machine learning paradigm where learning is done across multiple decentralized devices holding local pieces of data. Federated learning yields inherent security and privacy concerns, whose study for bandit problems recently started in the machine learning community e.g., [DP20]. To begin with, we are interested in investigating whether our cryptography-based secure and distributed bandit protocols could also make sense in a federated learning

setting. Then, we plan to rely on cryptography for securing other types of federated sequential learning algorithms. We aim at building protocols with well-understood theoretical properties, as well as open source system prototypes.

Interactive data exploration. This research direction is related to Chapter 4, to my PhD work on query specification for non-expert users, and also implies the study of new topics such as data exploration. The idea of finding a synergy between the aforementioned topics is quite natural since I recently acquired security and machine learning skills on which I could rely to develop algorithms for learning complex relational queries that help a non-expert user to explore some dataset that potentially contains sensitive attributes. More precisely, we are interested by the setting where some non-expert user wants to explore some large dataset by interactively generating a dashboard summarizing the dataset. By dashboard, we mean a collection of panels, whereas by panel we mean statistics that provide insights on a region in the data e.g., the result of a grouping and aggregation query, or the result of a clustering algorithm. Since the number of possible panels that one could express over a dataset is very large, we need an efficient way to propose meaningful panels to the user. We plan to cast the problem of proposing panels to the user as a multi-armed bandit problem, in order to find a good trade-off between *exploiting* panels close to what the user has already on her dashboard vs *exploring* completely different panels in order to dig into currently unexplored pieces of data. From a different point of view, if we outsource some part of the interactive dashboard generation to the cloud, we may choose to outsource encrypted data for the sensitive attributes and perform the aggregates directly in the encrypted domain. To sum up, we aim at building a *secure by design* system that relies on bandits in order to help non-expert users to explore data. We plan to evaluate the usability of our system with real users from domains e.g., behavior analytics or personalized medicine.

Chapter 7

References

Section 7.1 contains my list of publications, whereas Section 7.2 contains the other references.

7.1 My Publications

Publications after my MCF recruitment

[CLLS20] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare. Secure Outsourcing of Multi-Armed Bandits. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 202–209, 2020. https://ieeexplore.ieee.org/document/9343228

[CDLS20] R. Ciucanu, A. Delabrouille, P. Lafourcade, and M. Soare. Secure Cumulative Reward Maximization in Linear Stochastic Bandits. In *International Conference on Provable and Practical Security (ProvSec)*, pages 257–277, 2020. https://doi.org/10.1007/978-3-030-62576-4_13

[CL20b] R. Ciucanu and P. Lafourcade. GOOSE: A Secure Framework for Graph Outsourcing and SPARQL Evaluation. In *IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, pages 347–366, 2020. https://doi.org/10.1007/978-3-030-49669-2_20, Talk at https://www.youtube.com/watch?v=ZhtpulFf3rs

[CL20a] R. Ciucanu and P. Lafourcade. Demonstration of GOOSE: A Secure Framework for Graph Outsourcing and SPARQL Evaluation. In *International Semantic Web Conference (ISWC) – Demo Track*, 2020. http://ceur-ws.org/Vol-2721/paper476.pdf

[CLLS19] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare. Secure Best Arm Identification in Multi-Armed Bandits. In *International Conference on Information Security Practice and Experience (ISPEC)*, pages 152–171, 2019. https://doi.org/10.1007/978-3-030-34339-2_9

[CGLY19a] R. Ciucanu, M. Giraud, P. Lafourcade, and L. Ye. Secure and Efficient Matrix Multiplication with MapReduce. In *International Joint Conference on e-Business and Telecommunications (ICETE) – Revised Selected Papers*, pages 132–156, 2019. https://doi.org/10.1007/978-3-030-52686-3_6

[CGLY19c] R. Ciucanu, M. Giraud, P. Lafourcade, and L. Ye. Secure Strassen-Winograd Matrix Multiplication with MapReduce. In *International Joint Conference on e-Business and Telecommunications (ICETE) – Volume 2: International Conference on Security and Cryptography (SECRYPT)*, pages 220–227, 2019. https://doi.org/10.5220/0007916302200227

[CGLY19b] R. Ciucanu, M. Giraud, P. Lafourcade, and L. Ye. Secure Intersection with MapReduce. In *International Joint Conference on e-Business and Telecommunications (ICETE) – Volume 2: International Conference on Security and Cryptography (SECRYPT)*, pages 236–243, 2019. https://doi.org/10.5220/0007918902360243

[BCG+18] X. Bultel, R. Ciucanu, M. Giraud, P. Lafourcade, and L. Ye. Secure Joins with MapReduce. In *International Symposium on Foundations and Practice of Security (FPS)*, pages 78–94, 2018. https://doi.org/10.1007/978-3-030-18419-3_6

[CGLY18] R. Ciucanu, M. Giraud, P. Lafourcade, and L. Ye. Secure Grouping and Aggregation with MapReduce. In *International Joint Conference on e-Business and Telecommunications (ICETE) – Volume 2: International Conference on Security and Cryptography (SECRYPT)*, pages 514–521, 2018. https://doi.org/10.5220/0006843805140521

[BCGL17] X. Bultel, R. Ciucanu, M. Giraud, and P. Lafourcade. Secure Matrix Multiplication with MapReduce. In *International Conference on Availability, Reliability and Security (ARES)*, pages 11:1–11:10, 2017. https://doi.org/10.1145/3098954.3098989

[ACM17a] K. Alami, R. Ciucanu, and E. Mephu Nguifo. EGG: A Framework for Generating Evolving RDF Graphs. In *International Semantic Web Conference (ISWC) – Demo Track*, 2017. https://iswc2017.semanticweb.org/paper-460/

[ACM17b] K. Alami, R. Ciucanu, and E. Mephu Nguifo. Synthetic Graph Generation from Finely-Tuned Temporal Constraints. In *International Workshop on Large-Scale Time Dependent Graphs (TDLSG) @ECML/PKDD*, pages 44–47, 2017. http://ceur-ws.org/Vol-1929/paper1.pdf

[BBC+17a] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. gMark: Schema-Driven Generation of Graphs and Queries. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 29(4):856–869, 2017. https://doi.org/10.1109/TKDE.2016.2633993

[BBC+17b] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. gMark: Schema-Driven Generation of Graphs and Queries. In *IEEE International Conference on Data Engineering (ICDE) – TKDE Poster Track*, pages 63–64, 2017. https://doi.org/10.1109/ICDE.2017.38

[BBC+16] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. Generating Flexible Workloads for Graph Databases. *International Conference on Very Large Data Bases (VLDB) – Demo Track*, 9(13):1457–1460, 2016. https://doi.org/10.14778/3007263.3007283

Publications before my MCF recruitment

[BCS16] A. Bonifati, R. Ciucanu, and S. Staworko. Learning Join Queries from User Examples. *ACM Transactions on Database Systems (TODS)*, 40(4):24(1–38), 2016. https://doi.org/10.1145/2818637

[SOC16] M. Schleich, D. Olteanu, and R. Ciucanu. Learning Linear Regression Models over Factorized Joins. In *ACM International Conference on Management of Data (SIGMOD)*, pages 3–18, 2016. https://doi.org/10.1145/2882903.2882939

[AGCM15] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. The iBench Integration Metadata Generator. *International Conference on Very Large Data Bases (VLDB)*, 9(3):108–119, 2015. https://doi.org/10.14778/2850583.2850586

[ACGM15] P. C. Arocena, R. Ciucanu, B. Glavic, and R. J. Miller. Gain Control over your Integration Evaluations. *International Conference on Very Large Data Bases (VLDB) – Demo Track*, 8(12):1960–1963, 2015. https://doi.org/10.14778/2824032.2824111

[BCL15b] A. Bonifati, R. Ciucanu, and A. Lemay. Learning Path Queries on Graph Databases. In *International Conference on Extending Database Technology (EDBT)*, pages 109–120, 2015. https://doi.org/10.5441/002/edbt.2015.11

[BCL15a] A. Bonifati, R. Ciucanu, and A. Lemay. Interactive Path Query Specification on Graph Databases. In *International Conference on Extending Database Technology (EDBT) – Demo Track*, pages 505–508, 2015. https://doi.org/10.5441/002/dbt.2015.44

[BCS15] I. Boneva, R. Ciucanu, and S. Staworko. Schemas for Unordered XML on a DIME. *Theory of Computing Systems (TOCS)*, 57(2):337–376, 2015. https://doi.org/10.1007/s00224-014-9593-1

[BBC15] I. Boneva, A. Bonifati, and R. Ciucanu. Graph Data Exchange with Target Constraints. In *International Workshop on Querying Graph Structured Data (GraphQ) @EDBT/ICDT*, pages 171–176, 2015. http://ceur-ws.org/Vol-1330/paper-28.pdf

[BCS14b] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive Join Query Inference with JIM. *International Conference on Very Large Data Bases (VLDB) – Demo Track*, 7(13):1541–1544, 2014. https://doi.org/10.14778/2733004.2733025

[BCS14a] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive Inference of Join Queries. In *International Conference on Extending Database Technology (EDBT)*, pages 451–462, 2014. https://doi.org/10.5441/002/edbt.2014.41

[BCLS14] A. Bonifati, R. Ciucanu, A. Lemay, and S. Staworko. A Paradigm for Learning Queries on Big Data. In *International Workshop on Bringing the Value of Big Data to Users (Data4U) @VLDB*, pages 7–12, 2014. https://doi.org/10.1145/2658840.2658842

[CS13] R. Ciucanu and S. Staworko. Learning Schemas for Unordered XML. In *International Symposium on Database Programming Languages (DBPL)*, 2013. https://arxiv.org/abs/1307.6348

[BCS13] I. Boneva, R. Ciucanu, and S. Staworko. Simple Schemas for Unordered XML. In *International Workshop on the Web and Databases (WebDB) @SIGMOD/PODS*, pages 13–18, 2013. http://webdb2013.lille.inria.fr/Paper%2030.pdf

[Ciu13] R. Ciucanu. Learning Queries for Relational, Semi-structured, and Graph Databases. In *SIGMOD/PODS PhD Symposium*, pages 19–24, 2013. https://doi.org/10.1145/2483574.2483576

PhD thesis

[Ciu15] Radu Ciucanu. *Cross-Model Queries and Schemas: Complexity and Learning (Requêtes et Schémas Hétérogènes : Complexité et Apprentissage)*. PhD thesis, Université Lille 1 / Inria Lille / CRIStAL, 2015. https://tel.archives-ouvertes.fr/tel-01182649

7.2 Other References

[ABM10] J.-Y. Audibert, S. Bubeck, and R. Munos. Best Arm Identification in Multi-Armed Bandits. In *Conference on Learning Theory (COLT)*, pages 41–53, 2010

[ACF02] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3):235–256, 2002

[AES01] Advanced Encryption Standard (AES). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2001. Federal Information Processing Standards Publication 197

[AHÖD14] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified Stress Testing of RDF Data Management Systems. In *International Semantic Web Conference (ISWC)*, pages 197–212, 2014

[ALC18] N. Aburawi, A. Lisitsa, and F. Coenen. Querying Encrypted Graph Databases. In *International Conference on Information Systems Security and Privacy (ICISSP)*, pages 447–451, 2018

[APS11] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári. Improved Algorithms for Linear Stochastic Bandits. In *Conference on Neural Information Processing Systems (NIPS)*, pages 2312–2320, 2011

[AtG+15] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and Implementation of the LogicBlox System. In *ACM International Conference on Management of Data (SIGMOD)*, pages 1371–1382, 2015

[Aue02] P. Auer. Using Confidence Bounds for Exploitation-Exploration Trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002

[BCLO09] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-Preserving Symmetric Encryption. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 224–241, 2009

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 1997

[BDPMÖ12] E.-O. Blass, R. Di Pietro, R. Molva, and M. Önen. PRISM - Privacy-Preserving Search in MapReduce. In *International Symposium on Privacy Enhancing Technologies Symposium (PETS)*, pages 180–200, 2012

[BFVY18] A. Bonifati, G. H. L. Fletcher, H. Voigt, and N. Yakovets. *Querying Graphs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018

[BMMP18] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Annual Cryptology Conference (CRYPTO)*, pages 483–512, 2018

[BMT20] A. Bonifati, W. Martens, and T. Timm. An Analytical Study of Large SPARQL Query Logs. *VLDB J.*, 29(2):655–679, 2020

[BO15] F. Baldimtsi and O. Ohrimenko. Sorting and Searching Behind the Curtain. In *Financial Cryptography*, pages 127–146, 2015

[BR94] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 92–111, 1994

[BS09] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009

[CBS15] S. Chu, M. Balazinska, and D. Suciu. From Theory to Practice: Efficient Join Query Evaluation in a Parallel Database System. In *International Conference on Management of Data (SIGMOD)*, pages 63–78, 2015

[CDN01] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 280–299, 2001

[CJP+11] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: a Database Service for the Cloud. In *Conference on Innovative Data Systems Research (CIDR)*, pages 235–240, 2011

[CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, pages 409–437, 2017

[DBRT18] R. Delanaux, A. Bonifati, M.-C. Rousset, and R. Thion. Query-Based Linked Data Anonymization. In *International Semantic Web Conference (ISWC)*, pages 530–546, 2018

[DDGS16] P. Derbeko, S. Dolev, E. Gudes, and S. Sharma. Security and Privacy Aspects in MapReduce on Clouds: A Survey. *Computer Science Review*, 20:1–28, 2016

[DG04] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004

[DLOP17] J.-G. Dumas, P. Lafourcade, J.-B. Orfila, and M. Puys. Dual Protocols for Private Multi-party Matrix Multiplication and Trust Computations. *Computers & Security*, 71:51–70, 2017

[DLS16] S. Dolev, Y. Li, and S. Sharma. Private and Secure Secret Shared MapReduce. In *IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, pages 151–160, 2016

[DP20] A. Dubey and A. Pentland. Differentially-Private Federated Linear Bandits. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020

[Dwo06] C. Dwork. Differential Privacy. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1–12, 2006

[EALP+15] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz. The LDBC Social Network Benchmark: Interactive Workload. In *ACM International Conference on Management of Data (SIGMOD)*, pages 619–630, 2015

[FKPS17] J. Fernández, S. Kirrane, A. Polleres, and S. Steyskal. Self-Enforcing Access Control for Encrypted RDF. In *Extended Semantic Web Conference (ESWC)*, pages 607–622, 2017

[FKPS18] J. Fernández, S. Kirrane, A. Polleres, and S. Simon. HDT_{crypt}: Compression and Encryption of RDF Datasets. *Semantic Web Journal*, 2018

[FNP04] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 1–19, 2004

[Gen09] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Symposium on Theory of Computing (STOC)*, pages 169–178, 2009

[GGL12] V. Gabillon, M. Ghavamzadeh, and A. Lazaric. Best Arm Identification: A Unified Approach to Fixed Budget and Fixed Confidence. In *Conference on Neural Information Processing Systems (NIPS)*, pages 3221–3229, 2012

[Gie05] M. Giereth. On Partial Encryption of RDF-Graphs. In *International Semantic Web Conference (ISWC)*, pages 308–322, 2005

[Gir19] M. Giraud. *Secure Distributed MapReduce Protocols: How to Have Privacy-Preserving Cloud Applications?* PhD thesis, Université Clermont Auvergne / LIMOS, 2019

[Gol04] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004

[GPH05] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005

[GUK18] P. Gajane, T. Urvoy, and E. Kaufmann. Corrupt Bandits for Preserving Local Privacy. In *International Conference on Algorithmic Learning Theory (ALT)*, pages 387–412, 2018

[GRGP01] K. Y. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2):133–151, 2001

[HEl] HElib. http://homenc.github.io/HElib/

[HK16] F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19:1–19:19, 2016

[KAMD13] S. Kirrane, A. Abdelrahman, A. Mileo, and S. Decker. Secure Manipulation of Linked Data. In *International Semantic Web Conference (ISWC)*, pages 248–263, 2013

[KCG16] E. Kaufmann, O. Cappé, and A. Garivier. On the Complexity of Best-Arm Identification in Multi-Armed Bandit Models. *Journal of Machine Learning Research*, 17:1:1–1:42, 2016

[KKWZ15] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia. *Learning Spark: Lightning-Fast Big Data Analysis*. O'Reilly, 2015

[KSAK13] A. Kasten, A. Scherp, F. Armknecht, and M. Krause. Towards Search on Encrypted Graph Data. In *PrivOn@ISWC*, 2013

[KSS13] P. Kohli, M. Salek, and G. Stoddard. A Fast Bandit Algorithm for Recommendation to Users With Heterogenous Tastes. In *AAAI Conference on Artificial Intelligence*, 2013

[KVd18] S. Kirrane, S. Villata, and M. d'Aquin. Privacy, Security and Policies: A Review of Problems and Solutions with Semantic Web Technologies. *Semantic Web*, 9(2):153–161, 2018

[Jen] Apache Jena. https://jena.apache.org/

[JGGL20] L. Jachiet, P. Genevès, N. Gesbert, and N. Layaïda. On the Optimization of Recursive Relational Queries: Application to Graph Queries. In *ACM International Conference on Management of Data (SIGMOD)*, pages 681–697, 2020

[LCLS10] L. Li, W. Chu, J. Langford, and R. E. Schapire. A Contextual-bandit Approach to Personalized News Article Recommendation. In *International Conference on World Wide Web (WWW)*, pages 661–670, 2010

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014

[LS20] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020

[MBC13] T. Mayberry, E.-O. Blass, and A. H. Chan. PIRMAP: Efficient Private Information Retrieval for MapReduce. In *Financial Cryptography*, pages 371–385, 2013

[MP16] M. Meimaris and G. Papastefanatos. The EvoGen Benchmark Suite for Evolving RDF Data. In *MEPDaW/LDQ@ESWC*, pages 20–35, 2016

[MT15] N. Mishra and A. Thakurta. (Nearly) Optimal Differentially Private Stochastic Multi-Arm Bandits. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 592–601, 2015

[Mun14] R. Munos. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. *Foundations and Trends in Machine Learning*, 7(1):1–129, 2014

[Neo] Neo4j. http://neo4j.com

[Pai99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, 1999

[Pat12] D. A. Patterson. For Better or Worse, Benchmarks Shape a Field: Technical Perspective. *Communications of the ACM*, 55(7):104, 2012

[PRZB11] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2011

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning - An Introduction*. MIT Press, 1998

[SEA] Microsoft SEAL. https://github.com/Microsoft/SEAL

[Sha79] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979

[SHLP09] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL Performance Benchmark. In *IEEE International Conference on Data Engineering (ICDE)*, pages 222–233, 2009

[SLM14] M. Soare, A. Lazaric, and R. Munos. Best-Arm Identification in Linear Bandits. In *Conference on Neural Information Processing Systems (NIPS)*, pages 828–836, 2014

[SPL15] K. Semertzidis, E. Pitoura, and K. Lillis. TimeReach: Historical Reachability Queries on Evolving Graphs. In *International Conference on Extending Database Technology (EDBT)*, pages 121–132, 2015

[SPZ16] S. Y. Shah, B. Paulovicks, and P. Zerfos. Data-at-Rest Security for Spark. In *IEEE International Conference on Big Data*, pages 1464–1473, 2016

[SS18] R. Shariff and O. Sheffet. Differentially Private Contextual Linear Bandits. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 4301–4311, 2018

[TB09] J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In *Extended Semantic Web Conference (ESWC)*, pages 308–322, 2009

[TD16] A. C. Y. Tossou and C. Dimitrakakis. Algorithms for Differentially Private Multi-Armed Bandits. In *AAAI Conference on Artificial Intelligence*, pages 2087–2093, 2016

[Tho33] W. R. Thompson. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 1933

[TNP15] Q.-C. To, B. Nguyen, and P. Pucheral. TrustedMR: A Trusted MapReduce System Based on Tamper Resistance Hardware. In *OTM Conferences – International Conference on Cooperative Information Systems (CoopIS)*, pages 38–56, 2015

[TNP16] Q.-C. To, B. Nguyen, and P. Pucheral. Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture. *ACM Transactions on Database Systems (TODS)*, 41(3):16:1–16:43, 2016

[VBN19] T. D. Vo-Huu, E.-O. Blass, and G. Noubir. EPiC: Efficient Privacy-Preserving Counting for MapReduce. *Computing*, 101(9):1265–1286, 2019

[vG13] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2013

[Vir] Virtuoso. http://virtuoso.openlinksw.com

[VMKK14] M. Valko, R. Munos, B. Kveton, and T. Kocák. Spectral Bandits for Smooth Graph Functions. In *International Conference on Machine Learning (ICML)*, pages 46–54, 2014