Distributed and Privacy-Preserving Personal Queries on Personal Clouds

Julien Loudet

université PARIS-SACLAY

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

NNT : 2019SACLV067

Thèse de doctorat

Distributed and Privacy-Preserving Personal Queries on Personal Clouds

Thèse de doctorat de l'Université Paris-Saclay
préparée à Université de Versailles Saint-Quentin-en-Yvelines

École doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Versailles, le 24/10/2019, par

JULIEN LOUDET

Composition du Jury :

Pierre SENS Professeur, Sorbonne Université (LiP6)	Président
David GROSS-AMBLARD Professeur, IRISA (Druid)	Rapporteur
Vincent ROCA Chargé de recherche, INRIA (Privatics)	Rapporteur
Aline CARNEIRO VIANA Chargée de recherche, INRIA (TRiBE)	Examinatrice
Luc BOUGANIM Directeur de recherche, INRIA (Petrus)	Directeur de thèse
Iulian SANDU-POPA Maître de conférences, UVSQ (Petrus)	Co-encadrant de thèse
Benjamin ANDRÉ Cozy Cloud	Invité

Acknowledgments

My first thanks go to my advisors without whom most of this work would not have been possible (or, at least, much less thorough): Luc and Iulian. These last three years taught me a lot, they were also both challenging and fulfilling, and you played an important part in me getting through them. I will particularly remember the numerous discussions involving non politically correct examples and I sincerely hope no one ever recorded us. . . !

This PhD would also have not been possible without the involvement of Cozy Cloud and more particularly Benjamin, its CEO: thank you for giving me this opportunity!

However important and gratifying achieving this PhD was, what I will remember the most from this period is definitively the people that were with me during these past three years: Paul — who showed me both Cozy Cloud and SMIS/Petrus, with whom I had lengthy (and gossipy) discussions in the parking lot, who reminded me what seriousness means; Riad — who, despite my french origins, accepts me as a "kho", who never backs down from anything, whose blood will always be green (one, two, three. . . !), who is the one and only spirit of the team; Dimitris — who can't quite help himself from pointing out all the greek words we use, who is (almost) always joyful and unaffected by events, who never complained whenever I couldn't stop talking about my personal life; Aydogan — whose mustachy dream I wish to never have but whose calves I envy; Zoé — who always gave me a listening and supportive ear, who valiantly and consistently refused to help me with my state-of-the-art (despite how interesting of an offer it is!); Robin — the tallest kid among us all, who understands why the Zelda games are the greatest, whose PhD will (maybe) go smoothly; Laurent — "ma chérie" whom I always enjoyed disturbing; François — our own (super) handsome data-scientist; Gaëlle and Matthew — my two favorites rosbeefs who always helped me correct my english in my darkest hours; the Petrus team: Athanasia, Emmanuelle, Guillaume, Julien, Ludovic, Moeen, Nicolas, Philippe, Poulmanogo, Razvan; all the "Cozy family": Aeris, Aurore, Brendan, Bruno, Caroline, Cédric (and Latifa), Céline, Christophe, Claire, Clochix, Drazik, Éric, Erwan, Fabien, Florent, Frédéric, Grégory, Gooz, Joël, Joseph, Luc, Lucas, Martin, Matthias, Matthieu, Maxime, Nicolas, Nina, Patrick, Pierrot, Pierre, Quentin, Rémi, Romain, Sébastien B., Sébastien N., Simon, Thomas, Yannick, Yannou; my entire family who heard me going on and on and on about the PhD and never complained too much (I'm looking at you Ma'!); the "extended" LSI family: Djibril, François, José, Jérémy, Mohammad, Selma, Vaiyee; and all my other friends I did not mention: Aurélien and Isaline, Évrim, Kévin, Raphaël, Juba, Sébastien C., Suchet, Sylvain, Thomas, Valentin and Katerina.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As technology keeps improving, more and more "everyday-tasks" are automated, ranging from autonomous hoovers taking care of cleaning our homes to our web mail creating events in our calendar to remind us of an incoming trip. From a data-oriented point of view this translates into the generation of more and more *structured personal information*: the layout of our home, the dates and location of our holidays. Taken as is, this statement is not frightening: as data are structured, they are more easily processed by machines which leans toward an even better integration of these new technologies in our lives. However, the situation is not idyllic and growing concerns for our privacy are formulated: most of the interactions with these new technologies are centralized by the same actors, giving them a deep insight into our lives. To illustrate this statement we can look at the numbers boasted by the "Internet Giants": Facebook announced in 2017 that it had 2 billion active users on its platform[1]; in 2019, Google's search engine accumulated more than 75% of the searches made on the Internet[2]; WeChat has more than 1 billion active users in 2019[3]; in 2017, 20 million units of the Amazon Echo and 7 million of the Google Home have been sold[4]. Moreover, this only a first step in the digitalization of our lives: autonomous cars, internet of things, home automation, ... soon we will constantly interact with our "smart" environment. Although, this is not intrinsically problematic, leaving all the management of these devices to few, extremely powerful actors is: they will *centralize* our personal data, opening up the way for profiling, misuse (i.e. not in the interest of the users) and simultaneously increasing the appeal of an attack on their infrastructure as a single breach leads to millions (if not billions) of information.

Fortunately, not everything is grim in the landscape of personal data: the most concrete recent event has been the adoption of the *General Data Protection Regulation* by the European Union which, for instance, forces actors manipulating personal data to state their intents and explicitly ask for consent. More importantly, this regulation gives users legal grounds to request a copy of their own data and even to require companies to delete them: it **empowers** the users.

Other, older, initiatives are also worth mentioning: OpenStreetMap[5] aims at creating and

[1] https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook/

[2] https://www.netmarketshare.com/search-engine-market-share.aspx

[3] https://www.statista.com/statistics/255778/number-of-active-wechat-messenger-accounts/

[4] https://www.geekwire.com/2017/amazon-leads-smart-speaker-race-20m-devices-sold-study-claims-google-gaining-ground/

[5] https://wiki.osmfoundation.org/wiki/Main_Page

providing free geographic data, such as street maps, to anyone; Framasoft[6] is a non-profit organization providing free, privacy-friendly, open source alternatives to some well-known (non-free and definitively not privacy-friendly) services; Qwant is a search engine that also relies on advertisements for its business model but does not tailor them based on the users' profiles. The initiative that concerns the most the work done in this thesis is the *Personal Cloud*: the objective is to give to the users a digital home that they fully control and in which they can import their personal data, use them, benefit from them, i.e. do as they see fit. Cozy Cloud is one of the companies developing this type of solution and it is in collaboration with them that this work took place. Indeed, Cozy Cloud proposes logically (if not physically) separated single-user instances, which renders multi-user applications complex to implement, even more so if the aim is to keep the data disseminated and while protecting the users' privacy. Hence, this work tries to give a first answer to this problem: how to build a fully-distributed, generic, privacy-preserving framework to query a network a Personal Clouds.

This work is thus at the crossroads of three "domains": *Personal Data Management Systems* (in which the Personal Cloud belongs), *Privacy-Preserving Techniques* and *Distributed Systems*.

Several variations of Personal Data Management Systems exist: *Online personal clouds*, *Zero-knowledge-based personal clouds*, *Home cloud software*, *Home cloud plugs*, *Tamper-resistant home cloud*. These solutions address different functionalities and, more importantly, consider different trust models: Online personal clouds legally and contractually commit to never use the data stored on their servers, Zero-knowledge-based solutions assume the provider to be untrustworthy and rely on the user to manage encryption keys, and Tamper-resistant solutions delegate the management of the keys to dedicated hardware.

When referring to Privacy-Preserving Techniques we encompass two major topics of research: encryption-based solutions and privacy-preserving data publishing. Both topics propose ways of manipulating data so as to minimize (or even nullify) the amount of sensitive information disclosed. More specifically, encryption-based solutions such as multi-party computation algorithms and specific encryption schemes (homomorphic, fully-homomorphic, functional to only name a few) advocate for computing on encrypted data, while privacy-preserving data publishing solutions like k-anonymity, l-diversity, or differential-privacy introduce a preliminary step that aims at anonymizing the data before they can be used.

Distributed Systems is a topic that has been extensively studied and proposes two approaches to manage nodes in a network: either with an *unstructured* or with a *structured* overlay. As the name implies, in an unstructured overlay nodes are not "strictly" organized and strategies like flooding, random walks or expanding-ring are employed to discover new peers or route messages. Napster, Gnutella and the Blockchain are examples of such overlay. In a structured overlay, nodes are organized in order to create efficient communication paths which alleviate the overall load of the network but put (slightly) more strain on the nodes. Distributed Hash Tables are the main representative of this approach: Chord, Content-Addressable Network, Kademlia, Pastry.

As we can see, each of these technologies tackle a different problematic and in a context that is neither similar nor necessarily compatible with ours. They thus possess limitations we have to take into account: privacy-preserving techniques based on encryption generally do not scale well, while privacy-preserving data publishing techniques are usually ad-hoc solutions that depend on the nature of the data manipulated, not all Personal Cloud variations offer

[6] https://framasoft.org

high security guarantees and, lastly, distributed systems focus on availability and scalability rather than privacy.

Selecting the appropriate tools and correctly combining them in order to build a generic, scalable, fully-distributed and privacy-preserving framework to query the Personal Clouds of the users is the first gap in the current knowledge this work addresses.

More precisely, by *generic* we mean that we do not want to limit the capabilities of our query framework (i.e. it should not depend on the nature of the data manipulated or the type of computation), by *scalable* we mean that the query framework should be able to handle a large and possible increasing number of nodes seamlessly (i.e. with a low and limited additional cost), by *fully-distributed* we mean that no central server should be actively participating in the execution of a query, and, finally, by *privacy-preserving* we mean that the query framework should protect the data exchanged and ensure the anonymity of the nodes participating.

In order to conduct this study we only make the following assumptions: each user possesses a dedicated device hosting her Personal Cloud instance, an instance is able to establish direct communications with its peers and each device contains a secure component. Chapter 2 gives a more in-depth view of this final assumption.

Our work also considers the following limitations: given that, to the best of our knowledge, we are the first to tackle this problematic in this specific context, we solely focus on the protection of the data exchanged and the anonymity of the participants. The information leakage linked to the expressiveness of the query results is orthogonal to this work.

Hence, to answer the exposed problematic, we make four main contributions, listed in the order they are introduced:

1. We propose a set of five requirements detailing the specifics any solution, evolving in the same environment as the one we describe, should respect. Each of these requirements deals with a specific aspect: preventing an attacker from influencing the query execution; preventing any node from concentrating information it does not own; splitting the execution in independent tasks; and protecting the identity of the participants as well as the content of their communications.

2. We propose *SEP2P*, a protocol that leverages the overlay organizing the network and a distributed random generation algorithm to enforce the first requirement: preventing an attacker from influencing the query execution. This protocol's main advantage resides in its ability to only require the participation of a limited set of nodes to achieve its objective.

3. We propose *DISPERS*, a protocol that applies the last three requirements to split and distribute the execution of a query. We define a query as a three parts structure, one of which restrains the participants by specifying relevance criteria. We supplement this selection by a *sampling* that not only enforces this limitation but does so while preserving, to some extent, the quality of the result and actually limiting the impact of a leakage.

   We then successively apply the requirements which leads to the definition of three distinct "actor" roles, one for each part of the query. These roles are complemented by additional protections to severely limit the scope of possible attacks.

4. We finally propose an in-depth security analysis and performance evaluation of the DISPERS protocol, as well as two proof-of-concepts. We notably show in the security analysis that DISPERS achieves the optimal leakage.

   The first proof-of-concept is an interactive graphical interface built for a demonstration session during which attendees can try to bypass the security mechanisms present in DISPERS, in order to better grasp our contribution. The second proof-of-concept takes the form of a degraded implementation of DISPERS for the French ANR PerSoCloud project. By adapting the requirements to a centralized context and bringing in secure trustworthy external servers, called *enclaves*, we achieve a similar separation of concerns than in DISPERS and bring community sharing capabilities to the Cozy Cloud service.

This thesis is organized in six chapters, starting with the introduction, the current chapter, in which we detail the general context, motivations and contributions.

Chapter 2 draws up a panorama of the different subjects related to the thesis: we first give a more precise definition of the Personal Cloud paradigm, we then continue with ways of securing the data it contains and finally study the different manners we can organize a fully-distributed system.

Chapter 3 introduces our first two contributions: the requirements our design follows and *SEP2P* our protocol for generating a verifiable random list of actors. An evaluation completes the description of the protocol where we assess its scalability, resilience and the impact of the different system parameters.

The design and evaluation of the *DISPERS* protocol are respectively given in Chapter 4 and 5. Chapter 4 first assesses the feasibility of producing a fully-distributed protocol in our context, and then progressively shapes our solution by introducing the different requirements in the design. Chapter 5 follows up on this design by evaluating its security guarantees and its performance. We then provide two proof-of-concepts implementation aiming at displaying its capabilities during a demonstration session and fulfilling a use-case in the French ANR PerSoCloud project.

Chapter 6 concludes this thesis by summarizing the contributions and giving some interesting directions for future work.

Chapter 2

Background Knowledge and Related Works

Contents

In this Chapter, we present the three main research areas related to our work: in Section 2.1 we discuss the different variations of the Personal Cloud starting with the more widely available online solutions and finishing up with self-hosted solutions hardened by tamper-resistant hardware. In Section 2.2, we describe the available means that exist to increase the security when performing data-oriented tasks: encryption, hardware-based protections and privacy-preserving data publishing. Finally, in Section 2.3 we discuss the different ways of organizing a fully-distributed network of nodes and explore some of the attacks these systems face as well as their specific countermeasures.

2.1 Personal Cloud Solutions

To give users the possibility to benefit from their data and to use them however they wish, they have to be able to store them in a convenient manner. More and more academia projects and industry led softwares propose credible solutions: Secure Personal Data Server[4], Open Personal Data Store[27], Cozy Cloud[24], Nextcloud[69]. Their goal is to offer a digital place owned and controlled by the user: her own *Personal Cloud*.

Hence, in this first section, we will provide a global picture of those solutions. Through this global picture, we want to give the reader an understanding of what those technologies offer, their architecture, their core functionalities, their trust model and their limitations. Our study follows the thorough survey realized in [6], which sorts Personal Cloud solutions in four categories: Online personal cloud solutions, Zero-knowledge-based personal clouds, Home clouds software and Home cloud plugs.

2.1.1 Online Personal Cloud Solutions

This type of Personal Cloud is the most well-represented in today's landscape: Cozy Cloud[24], Digi.me[28], Meeco[59], Nextcloud[69], BitsAbout.me[14], and even governmental programs like MyData[67] and MesInfos[62]. These different solutions help users gather their digital data at the same place, in a usable format and thus make possible cross-computations of data that are usually isolated. They claim to prohibit any secondary usage not specifically stated in their terms and to never disclose anything to third parties unless asked for by the users themselves.

Features. Online personal clouds typically offer three features: *data collectors*, *cross-data computations services* and *trusted data storage*.

Data collectors offer users a way to automatically fetch data coming from online services. At predefined time intervals, they connect to the online services, check if a new information is available and download it to the personal cloud if so. Cozy Cloud offers the "harvest" applications for that purpose, Digi.me proposes a catalog of connectors, and BitsAbout.Me focuses more on web activity trails.

Cross-data computations services are made possible because data that are normally scattered across the, closed, databases of service providers are now located at the same place. Online personal clouds can then leverage these different sources to perform computations — instead of doing the computation at each service provider (supposing this is a possibility). For instance, Digi.me offers a transversal data search, in Cozy Cloud applications discuss with the data system which exposes a number of "doctypes" corresponding to the different data it stores, Meeco organizes the data in the form of web-tiles and life-tiles which respectively consists in user-defined goals or activities on certain websites and user-defined data sets gathering specific files or personal information.

The trusted data storage feature corresponds to the logical (if not physical) separation of users' data within the cloud provider's infrastructure. Users can only access and perform computations on their own data. Expert users can also choose, when applicable, to host their instance on their own devices providing another separation. For example, in BitsAbout.Me, a *Personal Data Store* is attributed to each user and stored encrypted. In Digi.me, the data are encrypted and stored where the user wants. The encryption keys are derived from the user's password and deleted from the server at the end of each session, thus requiring the user to log in before they can be deciphered and exploited.

Trust model. Online personal cloud solutions make strong privacy promises in order to gain the trust of their users. They ensure that they will never observe, exploit or disclose data (unless asked for by the user). Three main arguments are put forward to nourish this trust: (i) the use of the security standards of authentication, communication and data encryption, (ii) legally binding contracts coupled with a respectful business model, and (iii) an accessible and/or audited code base.

The first argument portrays a safe environment: accessing or storing data is done securely, users can trust the platform, it respect the industry standards and thus no-one can snoop in on what you are doing.

The second argument dons a more formal appearance: online personal cloud providers and users are bound by a contract where all the treatments operated on the data are explicitly stated, plus, because their business model is different, it is not in their financial interest to deviate from it. Combined, these elements argue in favor of an environment where there is no reason to spy on the users, proposing de facto a data safe haven.

The final argument is there to say that either anybody (including security experts) can check or that security experts have checked the software. This means that any malicious code can be rapidly detected, pleading once again in favor of a trustable platform that can be verified.

Online personal cloud solutions cover similar functionalities: the collect of personal data, an individualized storage, and the capacity to associate data that would have otherwise been kept isolated. To gain the users' trust they make strong promises backed by legal commitments, a business model that is not data-oriented and good security practices.

However, although these promises are genuine, the extent to which they are actually binding is unclear. The security hypothesis on which these promises rely are also extremely strong: they assume that the cloud provider and the employees are honest and that the code is trusted. Moreover, as these solutions are centralized, if a leakage were to happen, the impact would be important as potentially all the data could be disclosed.

To reduce the extent of the promises and the strong security hypothesis, other solutions rely on encryption and a *zero-knowledge* approach.

2.1.2 Zero-Knowledge-Based Personal Clouds

Zero-knowledge-based personal clouds mainly possess architectural variations of Online personal cloud solutions. They consider that the service and cloud providers can be malicious and ensure that no one, except the owner, can access the raw content. To do so, they encrypt the data before sending it to the servers and they provide secure way to access them. SpiderOak through its Share product[88] or Sync[92] are such examples.

Features. They focus on two features: *secure storage* and *secure backup*.

To provide secure storage, the data are stored encrypted on the cloud and the users inherit the responsibility to store and manage the encryption keys. These keys are usually derived from the users' password to ease the management. For example, with Spideroak the cloud provider only knows the encrypted data, with Sync the encrypted keys are derived from the user's password, and in Mydex the system is separated into two parts, the client side which manages the keys during the duration of the session and the back-end storing the encrypted files.

Secure backup provides mostly recovery options and revision capabilities to retrieve a file after a malicious or unintentional loss.

Trust model. Three threats are considered: (i) an attacker compromises the cloud provider, (ii) the cloud provider uses the data in a way that was not intended and (iii) the client device failed or was corrupted. Encrypting the data on the servers addresses the first two threats while the secure backup feature prevents a device failure.

Zero-knowledge-based solutions offer a more secure approach as the data are not stored in clear. However, they offer a limited set of features as the encrypted data cannot be processed server-side. Another consequence of delegating the key management to the user and to a client-side application is that it actually weakens the security model: users' devices are more easily infected.

At the same time, keeping the data at the users' side permits more computations as they can be processed in their raw format and if the users' devices leverage secure hardware the same security guarantees (as with Zero-knowledge-based solutions) can be provided: *Home-cloud softwares* and *Home-cloud plugs*, which we detail next, respectively provide these elements.

2.1.3 Home-Cloud Softwares

Home-clouds (softwares or plugs) keep the data at the extremities of the network, as close as possible to the user and to the device that produced them. One of the main rationale is to effectively prevent any centralization.

More particularly, Home-cloud softwares mainly focus on how to properly share and monitor the users' data, to make sure that the privacy models they propose are respected. OpenPDS[27] and DataBox[38] are prominent examples of Home-cloud softwares.

Features. Home-cloud softwares offers three core functionalities: *trusted storage*, *cross-computations* and *data dissemination*.

Trusted storage is achieved by delegating the storage to the users' devices: OpenPDS users accumulate data about themselves on their personal devices (smartphone and/or computer) and then explore it through a privacy-preserving framework; DataBox separates the storage on different devices, called "stores", where each store corresponds to a user's device and is responsible for a specific type of data.

Cross-computations in OpenPDS are done through a query-answering system called *Safe Answer* that minimize the information disclosed by only answering precise questions and thus not revealing the complete data set. DataBox creates new data stores that hold the results of the queries and that are made available to third parties.

Data dissemination is a consequence of their query models: in both cases the users can better control and understand which data are sent and to which third-parties. Users choose how they disseminate their information.

Trust model. Their first assumption is that the devices which store the users' data are trusted. Their second is to assume that the different pieces of software they provide is trustworthy: the data system, the audit functionalities, the query framework. DataBox additionally advocates for the use of containers to isolate these different components to not solely rely on this consideration.

Unfortunately, no solid proofs are given to back up the later assumption: no formal security guarantees exist to demonstrate that the different elements respect the users' privacy.

Hence, Home-cloud softwares focus on providing querying and sharing capabilities that respect the users' privacy. To do so they store the data as close as possible to the users and provide them with means to monitor and manage how their data are disseminated. However, to provide these capabilities they assume that the devices and their software are trustworthy, which represents a strong hypothesis.

2.1.4 Home-Cloud Plugs

Home-cloud plugs differ from their purely software variant because they also provide dedicated hardware on which to host the Home-cloud software they developed.

Within the landscape of existing Home-cloud plugs we can differentiate two categories: those leveraging tamper-resistant hardware and those which do not. We start by describing the latter as its features and trust model are closer to Home-cloud softwares than the former.

Without Tamper-Resistant Hardware

The discontinued Lima and Helixee[70] projects were prime examples of the Home-cloud plugs. They sold dedicated hardware on which users could "self-host": they install a Home-cloud software on it in order to store, manage and benefit from their personal data.

Features. The main functionalities are *trusted storage and backup*. The data are stored encrypted locally and made accessible the users' devices by the plug which holds the encryption keys. Contacting the plug can be achieved by through a central DNS server which stores the IP addresses associated to each plug. It can also act as a secondary backup service: an encrypted archive is sent to the central server for a later recovery.

Trust model. Similar to the Home-cloud softwares, the main benefit of the Home-cloud plugs is the absence of a central server storing all the data. However, like with Home-cloud softwares, a strong assumption is made regarding the dedicated hardware and the software that runs on it: no formal proof is given regarding its security or its respect of the users' privacy, it has to be trusted. Although the attack surface is reduced because, normally, only authorized applications can run on the hardware, the lack of guarantees is problematic.

Hence, Home-cloud plugs, like their software equivalent, focus on providing trusted storage and backup capabilities as well as an eased accessibility from the different devices of the user. The lack of formal proof concerning the security provided by the hardware and software combination remains problematic even though the attack surface is constrained.

Leveraging tamper-resistant hardware to secure all the data-oriented operations is a way to provide concrete guarantees to Personal Cloud.

With Tamper-Resistant Hardware

Enhancing Home-cloud plugs with tamper-resistant hardware is a proposal that comes from research projects with the Personal Data Server[4] and Trusted Cells[7] as examples. Their approach is to embed a minimal Trusted Computing Base (TCB) that acts as a database management system within the secure element of the device, in order to form a decentralized and secured data platform. Because of its strong security guarantees, the computing power of the tamper-resistant hardware is usually limited.

Features. Three functionalities are showcased: *secure storage*, *secure cross-computation* and *secure distributed computations*.

Providing secure storage and cross-computations capabilities is relatively straight-forward: as the database management system is embedded inside the secure element, it inherits the security properties and can thus perform secure cross-computation. Encrypting the data on the device is the next step to completely securing the storage.

Making secure distributed computations is achieved by relying on a untrusted central server that possesses a much higher computing power compared to the tamper-resistant hardware. The data are first sanitized (encrypted or anonymized) and then transferred to the server which then realizes the computation or part of it. [98, 94] are examples of algorithms leveraging this *hybrid* architecture.

Trust model. The devices holding the personal data of the users can be trusted as, because they inherit the tamper-resistance of the hardware, software and hardware attacks are rendered extremely difficult. Furthermore, as the embedded database management system is relatively simple (due to the limited computing capabilities of the hardware), its administration also is and can thus be done by the users themselves.

In this context, most of the threats come from the supporting infrastructure (the elements directly around the secure element or the central untrusted server): it is considered as an adversary with *weakly malicious intents*[13], meaning that it can deviate from the proposed execution plan in order to learn sensitive information if and only if it cannot be detected.

Hence, associating a tamper-resistant hardware to a Home-cloud plug secures all the operations performed by a Home-cloud plug. However, because of the limited computing power of the secure element, it is impossible to enjoy a full range of applications without relying on an untrusted support server. Additionally, to formally prove the database management system, its design must be minimalist and, as such, extending it later on proves to be difficult.

2.1.5 Conclusion

All Personal Clouds variations offer to store the personal data of the users on a trusted storage: raw or encrypted on the servers of a cloud provider, on the users' devices, or on dedicated hardware. Depending on what can be done on this storage, i.e. depending on if it has the capacity to operate on them, additional features can be proposed: cross-computations, data dissemination, trusted backup, distributed computations.

Except for the last variant of Personal Cloud we described, strong assumptions regarding the trustworthiness of the platform are made: the devices or the different softwares, all are supposed to be trusted. Taking into account the recent breaches and attacks orchestrated against major corporations (e.g. Equifax in 2017, Marriott in 2018, or Capital One in 2019), solely relying on these assumptions is insufficient — especially more so if the data are centralized.

A Home-cloud plug combined with tamper-resistant hardware is the most adapted variation to our use-case: the data are close to the users and effectively decentralized, plus the secure element and the Trusting Computing Base form a provably secure and thus trustable combination. It is also the only variation that considers distributed computations, although while relying on a central untrusted support server, which is the focus of this work. Hence, in the remaining we consider that each user possesses a Home-cloud plug augmented with a tamper-resistant hardware.

The main reason for delegating part of the distributed computations is the lack of computing power of the considered secure element. As we will see in the next section, not only are there other ways of performing distributed computations, but not all secure elements are severely limited but also widely available.

2.2 Means for an Increased Security in Data-Oriented Tasks

There are three main options to perform data-oriented task in a secure manner: as we just saw we can rely on secure hardware, we can also compute over encrypted data, or we can first "sanitize" the data and only make the sanitized data available.

We start this section by presenting two widely available *Trusted Execution Environments* — Intel SGX and ARM TrustZone — that can be used as tamper-resistant hardware. We then describe existing work relying on tamper-resistant hardware to perform distributed computations. We follow up with encryption-based solutions: Functional Encryption, Multi-Party Computations and Homomorphic Encryption. And we lastly explore *Privacy-Preserving Data Publishing* options: k-anonymity, l-diversity, t-closeness, and ϵ-differential privacy.

2.2.1 Trusted Execution Environment

Secure Hardware (SHW) solutions are varied and widely used: the chips embedded in our credit cards, the SIM in our phones, some hard-drives, identity cards or passports. Their main feature is to securely store sensitive information but they can also provide some physical protection or use cryptographic primitives for various usages. Unfortunately, as most of those were developed for very specific use-cases, they are only appropriate for predefined scenarios. Hence, what can we use in combination with a PDMS? Quite obviously, the SHW should have some computing capabilities: if it only provides secure storage then, when doing distributed computations, the exchanged data would need to remain encrypted which would force us to use Homomorphic Encryption which we now know to be impractical. However, this constraint is not enough: smart cards do have computing capabilities but they are extremely limited and clearly not enough to permit generic computations efficiently. Thus, in addition to having computing capabilities, those capabilities should not be too restricted so as to not hinder the range of possible distributed computations.

Considering these elements, a certain class of SHW appears to match our needs: *Trusted Execution Environments (TEE)*. [80] gives the following definition of a TEE:

> **Trusted Execution Environment (TEE)** is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the run time states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and run time states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting back door security flaws are not possible.

Hence, a TEE is a hardware that provides *isolation*, from a potentially corrupted host system, *attestation*, i.e. proof that it runs the code it is supposed to, and *tamper-resistance*,

to both physical and software attacks. In the following we take a look at the two most widely available TEEs: *TrustZone* developed by AMD and *Secure Guard eXtension* developed by Intel.

TrustZone

TrustZone has recently been gaining traction both in academia and industry[73] with projects such as Android's Keystore[8], Open Portable TEE[51], Open-TEE[58] or TrustICE[90]. This increase in the interest is mainly due to the widespread adoption of devices embedding TrustZone compatible ARM processors: several billion mobile devices and more than half of the Internet of Things[73].

TrustZone's architecture can be summed up by the existence of two "worlds": a *secure* and a *normal* one. The *secure* world is isolated at the hardware level from the *normal* world: the memory is partitioned into secure and non-secure sections, processors change state depending on the execution, special registers are protected and can only be accessed by processors functioning in the secure state. To operate this "world transition" two strategies exist, depending on the generation of the processors: Cortex-A processors possess an extra processor mode that handles this task, the *monitor mode*; Cortex-M processors directly do this switch in exception handling code. This second option induces faster context switch and a lower power-consumption.

Applications using the secure world as a TEE can be separated into two, according to the architecture they put in place[73]: *TEE service* or *TEE kernel*.
A *TEE service* implements a single specific function and does not need guidance to manage its memory or cross-world communications. Each service is deployed on a single device so as to prevent any interference. DroidVault[49] and the Android Key Store[8] are representatives of this architecture: the first offers a trusted storage via a data protection manager that executes in the secure world, the second provides a container to secure cryptographic keys that can only be unlocked in the secure world.
A *TEE kernel* is much like a conductor: it implements a basic set of OS functions and orchestrates several TEE instances by managing their memory in the secure world, handling their communications or even providing them with APIs. Samsung Knox[83] and TrustICE[90] are examples of such architecture: they respectively provide secure containers or isolated computing environments in which "normal" applications are either executed or can access a restricted part of the storage and memory.

SGX: Software Guard EXtensions

The Intel Software Guard Extensions (SGX) technology was first introduced in 2013[5]. The objective is to give service providers and data holders the guarantee that their secrets and/or sensitive content are protected, to let them know which software is using them and in which environment. To do so, SGX generate protected software containers, called *enclaves*, that are isolated from the operating system and the hypervisor. This material isolation is notably ensured through encryption of part of the memory and the usage of specific parts of the processor (e.g. "measurement" registers), both of which provides confidentiality and integrity protections to the data. Several enclaves can be launched on the same platform, for example one for each version of the same software, and are all isolated from each other.

In addition to being able to access hardware-based security mechanisms, SGX equips

enclaves with two equally important features: *attestation* and *sealing*.
The *attestation* mechanism is there to provide a proof that a specific software is securely running within an enclave. An attestation can be "local", i.e. used between two enclaves running on the same platform, or "remote", i.e. for a third party outside the platform.
The *sealing* mechanism is used to persist the sensitive data once the enclave is destroyed. Indeed, due to their nature, special precautions must be taken to store them outside of the secure environment. SGX enclaves have access to *Sealing Keys* that they can use to encrypt and integrity-protect them. The sealing can be bound to several enclaves through the use of a Security Version Number: all enclaves possessing this number can unseal the data.

The scientific community has been extremely prolific with regard to the possible use-cases for SGX: hardening already existing technology[16, 41], revisiting others[64, 74], or even encrypting[33].

Weaknesses have also been highlighted by recent works[100, 25, 40]: for example, the foreshadow attack[100] showcased that it is possible to retrieve sensitive information from inside the enclave by speculating on the way it executes a program.

Trusted Execution Environments, through their two main representatives, offer a viable and widely available option to secure the data at the user's side. Hence, with little to no effort, users could be paired with a **secure** PDMS: an ARM processor with TrustZone or an Intel one with SGX. We make no further assumption regarding the type of TEE they use and we do not require any additional security component or mechanism: as long as it provides *isolation* and *tamper-resistance* to software and physical attacks, any TEE is suitable.

Hence, each user is equipped with a secure PDMS, with which we can ensure a global and equivalent level of security. We can then safely make them exchange sensitive information, effectively creating a fully distributed peer-to-peer system.

2.2.2 Secure Hardware Based Distributed Computations

PAMPAS[94] and the work by To et al.[97, 98] both leverage a decentralized network of nodes reinforced by tamper-resistant hardware to perform privacy-preserving computations.

PAMPAS: Privacy Aware Mobile Participatory Sensing Using Secure Probes

PAMPAS[94] objective is to perform privacy-aware mobile participatory sensing to monitor, for instance, urban activities such as noise, traffic, or air pollution. To do so they rely on a hybrid architecture composed of a *supporting server infrastructure (SSI)* and *secure probes (SP)*. The role of the SSI is to coordinate the communications and computations between the probes. A secure probe can be any portable device comprising a tamper-resistant secure element: a phone with a new generation Sim card for example. This secure element is considered to offer a high level of security and is thus assumed to be trusted (i.e. inviolable). This assumption also comes at a price: the secure element usually has low power CPU and a tiny RAM, effectively preventing any intensive operation.

The mobile participatory sensing is performed following a three phases protocol: (i) the SSI collects all the encrypted updates sent by the probes, (ii) the SSI starts a processing period during which a small subset of probes are randomly selected and are tasked to perform a partial aggregation of the updates (after decrypting them), (iii) the partial aggregate are sent to the querier which compute the final aggregation.

A private symmetric key is shared between all the probes and stored within the secure element (to prevent any malicious user or the SSI from accessing it), so that only the probes can decrypt the updates.

This approach yields several benefits: delegating the private key management to the secure element ensures that only these access the raw data of other users, relying on the SSI to coordinate the computations frees the secure elements from the task of maintaining a peer-to-peer overlay, and, except for intensive task, using secure elements yields no restriction on the computations that can be performed.

Nevertheless, having only one private key to encrypt (although it is changed regularly and propagated using asymmetric encryption) represents a single point of failure: if a single secure element is compromised then all the information can be leaked. The SSI worsens this situation as once it possesses the private symmetric keys it can decrypt all updates and even attribute them to specific probes.

Privacy-Preserving Query Execution on a Secure Decentralized Architecture

In [98, 97] their objective is to show that global computation and privacy protection are compatible concepts. They assume that the network is comprised of Trusted Cells [7] nodes — personal clouds that are secured by a secure hardware — and they want to be able to compute SQL-like queries on these, with a focus on *joins* and *aggregates* queries, while respecting the privacy of the users.

For this they assume that each personal cloud is a *Trusted Data Store (TDS)*: they offer high security, low availability, modest computing resources, they all share the same database scheme, are considered honest and the secure hardware inviolable. As with PAMPAS, they also introduce an honest-but-curious *supporting server infrastructure (SSI)* in order to execute a query.

The generic protocol for executing a query has three phases: a *collection phase*, an *aggregation phase* (optional) and a *filtering phase*.

During the collection phase, the Querier first puts up its encrypted query on the SSI. The TDS nodes download the query and either reply with dummy data or their local result if they have or can produce one (an access control operation is first performed). The local results, dummies or not, are encrypted using probabilistic encryption based on a shared symmetric key k_2 to prevent the SSI from conducting frequency-based attacks on the cyphertext. The query itself is encrypted using a shared symmetric key k_1. The key k_1 is known to all the TDS and the Querier, while k_2 is only known to all the TDS.

During the aggregation phase, the SSI partitions the local results so TDS nodes — not necessarily the ones that participated — can remove the dummy results and perform partial aggregations iteratively until all results have been aggregated, leading to a Covering Result. Partial results are encrypted using a shared symmetric key k_2 known to all TDS except for the Querier and the SSI.

Different ways of performing this phase are considered: instead of performing probabilistic encryption to hide the results, a combination of symmetric encryption and noise can be employed to attain the same result. Indeed, in both cases the SSI cannot conduct frequency-based attacks but in the latter it can help sort the results and speed up the aggregation process as it can group the results based on the, deterministic, cyphertext.

During the filtering phase, the SSI partitions once again the Covering Result so that TDS nodes can filter out either the dummy results (if no aggregation phase was needed), or removing the unwanted groups (for instance those that do not satisfy the `HAVING` clause of a query). At the end of this phase the SSI informs the Querier that the final result is available.

Executing distributed queries leveraging this hybrid architecture shows that it is possible to achieve a generic privacy-preserving query framework. However, similar to PAMPAS, the security hypothesis are strong: if a single TDS node is compromised, the full set of encryption key is divulged and the users' data are all at risk. If the SSI colludes with this single corrupted TDS node then every data of every query execution are disclosed, completely exposing the users.

Another way to perform distributed computations that notably solves this key management problem is to use encryption-based solutions.

2.2.3 Encryption-Based Solutions

There are three main encryption-based solutions to perform, generic, distributed computations: *Functional Encryption* that proposes an encryption scheme that only reveals the result of a specific function, *Multi-Party Computations* that proposes ad-hoc protocols and *Homomorphic Encryption* that can perform any computation directly on encrypted data.

Functional Encryption

Functional Encryption (FE)[15] allows fine-grained access control and selectivity in the processing of the encrypted data. The objective of a FE scheme is to provide a way to compute a specific function f over some encrypted data c and to only reveal $f(x)$, where x is the underlying plaintext associated to c. To achieve this, a public key and master secret key are first generated, then a secret key, sk_f, for the function f is derived from the master key, and the plaintext is encrypted using the public key. Using sk_f, f can be applied over the cyphertext and yields $f(x)$. This last operation is called *decryption*.

Although promising, Functional Encryption has two main drawbacks: (i) a different secret key must be computed for each function, making a complex processing impractical; and (ii) the data have to be encrypted with the same key, and thus the same party, the one possessing the public key. These elements prevent all distributed applications where the participants do not trust each other.

Multi-Input Functional Encryption[35] or Decentralized Multi-Client Functional Encryption[20] try to address these limitations but these solutions are not perfect either: their protection is either bound to the number of information the adversary has access to, or they are simply not generic.

Hence, being an active research topic and considering its properties, Functional Encryption will be an interesting approach to do distributed computations once these limitations are alleviated.

Multi-Party Computation

Introduced by Yao in 1982 with its, now, famous question — How can two millionaires know who is richer without disclosing their individual wealth to each other? — *Multi-Party Computation (MPC)* provides means to compute over data (not necessarily encrypted) and only

reveal the final answer to the participants. In other words, no participant knows more than its input and the final output.

Two model paradigms exist in the literature to solve MPC problems: the *ideal model* that supposes that there is at least one trusted third-parties among the participants and the *real model* that makes no such assumption. Considering the objective of this work, only solutions following the real model are applicable.

There are mainly three types of solutions to MPC problems[81, 85]: *anonymization*, *randomization* and *cryptography*.

Anonymization relies on one or several trusted third parties to hide the identities of the participants. It thus takes after the *ideal model* which we discarded.

Randomization consists in adding random values, "noise", to hide the data. As an illustration, the following protocol is a trivial way of computing a secure sum using randomization: (i) an initiator is first elected among the participants; (ii) this initiator adds a random number to her input and transmits the sum to the next party; (iii) the next party then adds her own number to this sum and sends the total to the next party; (iv) this procedure is repeated until all parties have contributed; (v) the complete sum is finally transmitted to the initiator who subtracts the random number and obtain the final result without revealing any individual input.

Cryptography techniques assemble basic cryptographic tools to construct a secure computation. Some of the most important cryptographic tools are: *Yao's millionaires problem* — comparing values without disclosing them, *Homomorphic encryption* — a type of encryption that preserves certain operation if they are performed on the cyphertext, *Oblivious Transfer* — a type of information transfer in which the sender does not know which information the receiver obtained, and *Private Matching* — an operation that lets two parties know the intersection of their data sets without revealing them.

Despite their interesting properties, those strategies are for the vast majority ad-hoc solutions for specific applications[23]: secure sum, secure set union, secure size of set intersection, secure scalar product, privacy-preserving statistical analysis. This makes it difficult to produce a generic framework that can be used in most situations.

More importantly, once an MPC protocol is devised in order to be computed it first has to be transformed into a boolean or arithmetic circuit which is a complex, error-prone and time-consuming task. Worst, its execution time can be extremely long, especially more so if the number of participants increases: in [52] they have managed to compute a single AES-256 circuit in 33ms but in a 2-party setting, with a highly-optimized circuit and relatively powerful hardware. These assumptions are not compatible with our desired setting.

Homomorphic Encryption

An encryption scheme is considered *homomorphic* if a certain operation is "preserved" when applied on the encrypted data: if $e(a)$ is the encrypted value of a and $e(b)$ of b, then, if $e(a) \otimes e(b) = e(a \otimes b)$, e is considered *homomorphic* over the operation $\otimes$.

Since the breakthrough by Gentry[34], homomorphic encryption (HE) schemes can be categorized into three groups[2]: *Partially Homomorphic Encryption (PHE)*, *Somewhat Homomorphic Encryption (SWHE)* and *Fully Homomorphic Encryption (FHE)*. As the first two groups can only perform a limited number of operations (PHE) or some operations but a limited number of times (SWHE), they cannot be applied in our context, since they would severely

hinder our capabilities to perform generic operations. Hence, let us study Fully Homomorphic Encryption.

The first FHE scheme developed by Gentry[34] opened a new research path for HE schemes that were until then either PHE or SWHE. An FHE scheme supports an unlimited number of operations for an unlimited number of times, meaning that any operation can theoretically be computed on encrypted data. Unfortunately, despite the major advancement this represents and the successive improvements made over this leading work, FHE has an excessive computational cost (in terms of time spent and resources required) which, in practice, makes it far from applicable: it took 4 minutes for the most efficient implementation[2] to evaluate the AES circuit homomorphically.[1]

Hence, state of the art encryption schemes and algorithms that stemmed from them offer a secure but impractical approach to data sharing, especially in a fully distributed context where no trusted third-party is available. Functional Encryption and Multi-Party Computations offer solutions to specific use-cases that are complex to extend, while Fully Homomorphic Encryption schemes overcome these limitations but at the price of an extremely high computational cost.

A way to alleviate this cost is to rely on Privacy-Preserving Data Publishing techniques that, instead of encrypting the data, first sanitize them before publishing.

2.2.4 Privacy-Preserving Data Publishing techniques

The basic idea behind Privacy-Preserving Data Publishing (PPDP) is to protect the private data, i.e. the user's sensitive information, by "distorting" the data that will be made public before its actual publication, at the expense of a loss of utility of the public data in the later processing stages.

This distortion operation is not a straightforward task: correctly performing it is highly dependent on the type of data to protect, on the background knowledge the attacker is presumed to have and on other publicly available complementary sources of information.

In this section we explore the four main approaches to PPDP: k-anonymity, l-diversity, t-closeness and ϵ-differential privacy.

k-Anonymity

Introduced by Samarati[82] and Sweeney[91], *k-anonymity*, aims at providing a measure concerning the "disclosure risk" of a given data set. The main idea is to compute the number of records that share the same values or subset of values: the more records there are and the less discriminative those values are. This analysis arose after the identification of the Governor of Massachusetts in a supposedly anonymized set of health records: the gender, zip code, date of birth and diagnosis of state employees were revealed, which, by itself, did not disclose any personally identifiable information but, once linked with the voters registration records, uniquely identified the Governor.

The following definition was given to such subset of values:

QUASI-IDENTIFIER. *A* quasi-identifier (QI) *is a set of non-sensitive attributes that, once linked with external data, can uniquely identify an individual in the general population.*

[1] The evaluation is: (i) a client sends the AES key k using FHE, i.e. FHE(k); (ii) the client uploads the data encrypted with AES, $\text{AES}_k(m)$; (iii) the server computes $\text{FHE}(\text{AES}_k(m))$; (iv) it homomorphically decrypts it to obtain FHE(m) which it can use to compute any homomorphic operation.

We also define an *equivalence class* as the set of records that share the same values for a QI. In the Massachusetts example, the quasi-identifier was the tuple `(zip code, birth date, sex)`.

Deriving a definition for k-anonymity is then straight-forward:

k-ANONYMITY. *A data set satisfies k-anonymity for a given QI if and only if each equivalence class for the QI appears with at least k occurrences in the data set.*

Age	Zip code	Occupation
≥ 20	65200	Student
≥ 20	65200	PhD Candidate
≥ 30	69000	Cook
≥ 30	69000	Nurse

Table 2.1: A 2-anonymous data set on (Age, Zip code)

Although it yields interesting properties on the data and offers, to some extent, certain guarantees on the privacy of the records, this method is not perfect. Indeed, very few data sets naturally possess this property for any QI, that is why the data are first *sanitized* to limit the possible QIs. Furthermore, as we will see next, in some particular cases the protection can be non-existent.

l-Diversity

l-diversity[56] stems from two weaknesses of the k-anonymity model: (i) an attacker can infer sensitive information if there is little diversity in the data set, and (ii) if attackers possess background-knowledge, k-anonymity cannot ensure privacy. Both those attacks can be illustrated by the following example:

Sex	Age	Condition
Male	3*	Cancer
Male	3*	Cancer
Female	2*	Heart Disease
Female	2*	Viral Infection

Table 2.2: Illustration of attacks on k-anonymity

Let us assume that this data set contains the "anonymized" records of the recently admitted patients of the nearby hospital. If a curious citizen knows that one of her male friend is in that list, she knows he has cancer since all male patients are treated for that condition. This is an illustration of attack (i), known as an *homogeneity attack*.

Suppose now, that the curious citizen also has a female friend in her twenties that was recently admitted. She knows that this friend is a talented athlete, subject to regular tests (included cardiac ones). Given this history she can infer with a relatively high probability that she was admitted for a viral infection. This illustrates attack (ii), called a *background-knowledge attack*.

To overcome those weaknesses, [56] proposed the l-diversity principle:

l-DIVERSITY PRINCIPLE. *To any set of values for a* quasi-identifier (QI) *should be associated, at least, l "well-represented" values for the sensitive attribute S.*

The notion of "representation" is left to an instantiation of the principle. A simplistic approach would be to require that each equivalence class has l distinct values for the sensitive attribute. Being simplistic, this approach has a quite obvious limitation: if a value is much more frequent than others, an attacker can do probabilistic inference attacks. [56] gives two additional and less trivial examples: *Entropy l-Diversity* and *Recursive (c, l)-Diversity.* As the name implies, *Entropy l-Diversity* views the representation of a sensitive value as its entropy and enforces that each should exceed a lower bound, $\log(l)$. The *Recursive (c, l)-Diversity* stipulates that the most frequent sensitive values should not appear too often and the least frequent ones no too infrequently: the number of times, r_1, the most frequent value appears should not overly exceed the sum of the less frequent ones, i.e. $r_1 < c \cdot \sum_{j=l}^{m} r_j$.

t-Closeness

t-closeness[48] proposes a model that, in the same vein as *l*-diversity for *k*-anonymity, aims at overcoming certain limitations of *l*-diversity. Indeed, *l*-diversity is (i) difficult and sometimes unnecessary to achieve and (ii) it does not always prevent attribute disclosure.

Limitation (i) is illustrated by a scenario where there are only two sensitive values and one is much more present than the other (e.g. HIV test results). Any record that has the less represented value is all the more sensitive, making a 2-diverse table extremely lossy as the upper bound of the number of equivalence classes would be equal to the number of records having the less represented value (to be unable to distinguish a record with one or the other value).

The same scenario can be used to explain (ii): any equivalence class containing the same number of sensitive values yields a probability of having the less represented value of 50%, which is much higher than in the overall population and thus presents a serious privacy risk.

To prevent those limitations, *t-closeness* considers the information gain relative to both the whole population and the current equivalence class:

t-CLOSENESS PRINCIPLE. *An equivalence class is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold t. A table is said to have t-closeness if all equivalence classes have t-closeness[48].*

To measure the distance between two distributions, the *Earth Mover's distance (EMD)*[79] is employed as it also captures, with a little bit of tweaking, the semantic distance between values.

ϵ-Differential Privacy

Proposed in [30], the core mechanism behind ϵ-*Differential Privacy* can be roughly summed up as: the addition, deletion or modification of a single record should have no significant effect (i.e. "less than ϵ") on the results of any analysis. To showcase this property, one could think of an insurance provider not being able to modify its attribution policy depending on whether or not the individuals are present in the database they consult to reach a decision.

A formal definition, given in [30], is:

ϵ-DIFFERENTIAL PRIVACY. *A randomized function K gives ϵ-differential privacy if for all data sets D_1 and D_2 differing on at most one element, and all $S \in Range(K)$,*

$$\Pr[K(D_1) \in S] \leq \exp(\epsilon) \times \Pr[K(D_2) \in S]$$

The probability is taken over the coin tosses of K.

This definition only provides a condition to respect, that is to ensure that the result of a randomized computation does not differ more than ϵ for two extremely close inputs. To achieve it, random noise is added to the query function — transforming it into a *randomized* function.

2.2.5 Conclusion

In this section we saw three ways of performing distributed computations while preserving (or, at least, trying to) the users' privacy.

Continuing on the previous section and the Home-Cloud plugs enhanced with tamper-resistant hardware, we started by detailing two widely available Trusted Execution Environments, ARM TrustZone and Intel SGX, that can be used as tamper-resistant hardware. We then saw two works that leverage a hybrid architecture composed of a supporting server infrastructure and a decentralized network of secure elements to perform privacy-preserving distributed computations. These solutions have two identical main drawbacks: (i) they are not fully decentralized as a support server is mandatory to coordinate the computation and (ii) their security model assume that the secure elements cannot be compromised, which leads them to share an encryption key that protects the exchanged data, thus forming a single point of failure.

We then explored encryption-based solutions: Functional Encryption that only discloses the output of a function while keeping the input data encrypted, Multi-Party Computation protocols that offer ad-hoc solutions to specific problems and Fully Homomorphic Encryption schemes that can compute any function on encrypted data (provided they are encrypted appropriately). These approaches also have limitations: Functional Encryption and Multi-Party Computations are not generic and hard to extend (if not hardly extensible), while the generic Fully Homomorphic Encryption schemes are far from scalable.

We lastly discussed Privacy-Preserving Data Publishing techniques: k-Anonymity, l-Diversity, t-Closeness, and ϵ-Differential Privacy. They offer ways of measuring the privacy leaks and means to contain them so that, in the end, the disclosed information is not enough to single out a user. Again, these strategies all have an intrinsic limitation: they assume knowledge on the entire distribution of the data (or at least to a sizable subset), an assumption that is relatively hard to materialize in a fully-distributed environment without a trusted third-party. Furthermore, making use of these solutions implies a trade-off in terms of utility: either we increase the privacy guarantees and loose quality or we reduce the privacy guarantees and increase quality.

Hence, no current solution satisfies our requirements of only relying on a fully-decentralized network of Home-Cloud plugs secured with tamper-resistant hardware and assuming that some can be corrupted. However, with the development, wide availability and reasonable computing power of recent Trusted Execution Environments, an architecture where the computations is distributed between the nodes and performed inside the secure element becomes feasible. This architecture is thus the one we choose to rely on in this work.

In order to efficiently use this network the first step is to properly organize it. Peer-to-Peer systems perfectly suit our needs: they leave the organization of the network to the peers.

2.3 Peer-to-Peer Systems

Peer-to-Peer (P2P) systems were popular at the beginning of the internet, at a time where hardware was not powerful enough to handle the data and interactions of a multitude of users. In this configuration, each node has the same importance and can play any role — an important aspect in a distributed computation.

The properties we are looking for in such system is the capacity to store and retrieve an information in an efficient manner, i.e. in as little communications as possible while storing as little information as possible. Indeed, as a *PDMS* is first and foremost the personal digital space of its owner, this primary features must remain unhindered.

In this section we will first take a look at the already existing solutions for organizing such network, from "unstructured" to "structured" techniques, which will lead us to *Distributed Hash Tables*, a solution that we will explore in more details. We will then explicit the typical threats faced by those systems: *Sybil attacks* — where an attacker floods the network with fake malicious nodes, and *Byzantine Fault Tolerance* — where, if an attacker controls a sufficient portion of the network, an honest node cannot function properly anymore.

2.3.1 Unstructured Peer-to-Peer systems

As stated previously, peer-to-peer systems are commonly divided into two categories: *unstructured* and *structured*.

As the name indicates, *unstructured* systems are not "strictly" organized: flooding, random walks, or expanding-ring time-to-live (TTL) techniques are used to query the graph of nodes composing the network. Indeed, as the nodes are not organized, they form a graph with no specific or predetermined shape thus rendering any globally efficient optimization impossible. Hence, in order to obtain an answer a sufficient portion of the network — if not all — must be interrogated.

Gnutella[95], Freenet[21] and the Blockchain[68] are iconic representatives of such system. Let us see how they operate.

Gnutella

The Gnutella protocol is rather simple: a node in the network is called a "servent" (a mix of server and client, "serv"–"ent", as nodes can play both roles) and has at its disposal five "descriptors": `Ping` — to discover new servents, `Pong` — to answer a `Ping` descriptor, `Query` — to search for some keywords, `QueryHit` — to answer a `Query` descriptor with relevant files names, and `Push` — to send a set of files to another servent (usually in reply to a `QueryHit` descriptor).

Being the first of its kind and because of its popularity and its simplistic design (and the resulting shortcomings), Gnutella has been extensively studied and several improvements have been proposed and implemented: Ultrapeers[87] to increase the routing performance, [37] measured key network metrics and showed that Gnutella is deeply heterogeneous, [19] describes a modified Gnutella, called *Gia*, that brings flow control, dynamic topology adaptation to render P2P system scalable.

Freenet

The first white paper describing Freenet dates back to 1999[2] and, to the day of this writing, Freenet's network is still active. Freenet[21] is a self-organizing and completely decentralized network that uses the free disk space of the participants to share files while, at the same time, protecting the privacy of its users, maintaining data integrity and adapting itself to the current usage patterns. As we will see later with the *Distributed Hash Tables*, Freenet uses *Globally Unique IDentifiers (GUID)* to assign a location to nodes and files in the network. In the case of files, this identifier is obtained by computing a secure hash of the file to be stored. The objective is to ensure uniqueness and integrity: identical files will receive the same *GUID* and once the file is received the *GUID* can be computed once again to check that the file was not altered. For nodes, the identifier is collaboratively generated following a "cryptographic protocol for shared random number generation that prevents any participant from biasing the result". With that number, the new node is assigned responsibility for a region of the key space.

Based on those *GUID*s, the routing, be it for storing or locating a file, is made possible: the *GUID* is computed and the nodes transfer to the closest[3] node they know the request until it either reaches its destination or fails. If the request is successful, the result is transferred back following the same path (in reverse direction) so that each node on the way can learn who is responsible for that *GUID* and enrich their routing information. The request fails if no contacted node manages the given *GUID* and the *Time-to-Live (TTL)* counter reaches zero. This counter is used to avoid flooding the network by killing requests after a certain point.

Freenet also requires each node to generate a pair of asymmetric keys before it can join the network. These keys are used to create "pointers" that indirectly reference files and ensure that only the creator can update these pointers.

Blockchain

The last example of unstructured peer-to-peer system is the *blockchain*. The blockchain technology came to light after the publication of [68], published under a pseudonym. The core idea is to solve the "double-spending problem" without relying on a trusted third-party. To do so, peers generate proof for each operation, the so-called *blocks*, which, once linked, form a chain tracing all operations: a "blockchain". This chain of proofs can then be checked by any peer and prevents any double-spending.

To generate a block, peers must *mine*. The mining operation is a costly CPU intensive task which consist in solving the following problem: given an input i, find a complementary data n (a `nonce`) such that $h(i + n)$ starts with δ leading zero bits — where δ is a constant fixed by the system, h a cryptographic hash function and $+$ a simple concatenation. Finding a solution requires an average work that is exponential with the number of zero bits required. Once a solution is found, the block, i.e. its input and the corresponding solution, is broadcast to all the peers in the network. As no solution is unique, several valid blocks, potentially attesting different inputs, can cohabit. To elect one, a longest chain policy is applied: each block references[4] a predecessor and the longest chain in the network is chosen as "correct". Proofs generated in that manner are deemed *proofs-of-work*.

[2] https://freenetproject.org/assets/papers/ddisrs.pdf

[3] The notion of "closeness" is left to implementation and is obtained by comparing two *GUID*s.

[4] Chosen by the miner.

Each block contains a hash of the previous block and a list of *transactions*. A transaction stipulates the previous and the current owner for the resource, r, it is attached to: (r) Alice $\rightarrow$ Bob. For later checks, each resource keeps the list of all its previous owners. Whenever a transaction occurs, it is broadcast to the entire network so that miners (possibly all the nodes in the network, hence the broadcast) can work on a next block.

The most famous blockchain is the one used by *Bitcoin*, named after the paper and the resource it proposes. In Bitcoin, whenever a block is mined, a certain quantity of resources is created and attributed to the peers that mined it. Other alternatives exist and rest upon the same principles such as Ethereum[31, 32], Litecoin[53], or Zcash[103, 104].

Due to its public nature, where all transactions are broadcast and recorded inside blocks that are also broadcast, it is with relative ease that one can make a compelling case against it for a privacy-respectful context. Nonetheless, a public record is not the only issue (even though it is still enough to disregard it): the creation of blocks demands resources thus turning PDMS into miners and potentially costing users a substantial amount of money, eventually preventing a mass adoption from the public.

2.3.2 Structured Peer-to-Peer Systems: Distributed Hash Tables

Organizing the overlay network mainly yields two advantages: firstly and most importantly it reduces the flooding of the entire network by creating efficient communication paths; secondly, it reduces the amount of additional "load" each node has to deal with to maintain a coherent overlay network (e.g. reduced routing tables, optimized redundancy, optimized storage).

To achieve it, a *Distributed Hash Table (DHT)* is usually created: unique keys are associated with data and nodes, these keys are then used to construct a structured graph and to assign data to nodes. In the remainder of this section we describe the most well-know DHT: Pastry[78], CAN[75], Kademlia[57] and Chord[89].

Pastry

Pastry[78] organizes the network following a circle. A random 128-bit unique numeric identifier, a `nodeId`, which ranges from 0 to $2^{128}-1$ is attributed to each node. The actual generation of those identifiers is left to implementation but it is assumed that it gives a uniform distribution (e.g. using a cryptographic hash function). The randomness in the attribution is to bring diversity to the set of neighbors each node has (e.g. in terms of geography, jurisdiction, ownership).

Pastry achieves on average a routing of $\lceil O(\log_{2^b}(N)) \rceil$ steps, where N is the number of nodes in the network and b is a system parameter — usually equal to 4. The routing is achieved by comparing the digits of the key of the destination to the key of the, successive, nodes. For example to route a query to 1234, it would go to any node matching $***4$ then $**34$ (e.g. $AE34$ matches $**34$), $*234$ and finally 1234. The routing also takes into account network locality by considering a proximity metric and always favoring the "closest" node.

To store a file in the network, a hash of the file's name and owner is computed and used as a `fileId`. The k nodes having the numerically closest `nodeId`s to the `fileId` will store the file.

To correctly forward look up queries each node maintains three sets of nodes: a *routing table*, a *neighborhood set* and a *leaf set*. The routing table contains $\lceil O(\log_{2^b}(N)) \rceil$ rows of $2^b - 1$ nodes, where the entries in row n refer to nodes that share the first n digits of the

current node's `nodeId` and not the $n+1$ digit. The value of b involves a trade-off: a higher value reduces the number of hops required to reach the destination at the cost of an increased size of the routing table. The neighborhood set regroups the M closest nodes, according to the proximity metric. This list helps at providing only "close" nodes in the routing table: it buffers the closest nodes, regardless of their `nodeId`, by regularly probing its neighbors' neighborhood sets. The leaf set contains L nodes divided into two groups: $|L|/2$ nodes that have the numerically closest larger `nodeId` and $|L|/2$ nodes that have the numerically closest smaller `nodeId`. This set ensures appropriating routing if the routing table fails at providing the next link in the chain of forwarding.

To join Pastry, a new node starts by computing its `nodeId` and then asks a node already belonging to the network a "join" query providing its `nodeId`. All the nodes on the path of that query, which ends once the closest node to its `nodeId` is reached, return their sets to the new node so that it can build its own.

When a node leaves the network, all the nodes update their different sets whenever the departure is detected. For the *leaf set*, the node with the largest `nodeId` on the side of the node that left is asked for its leaf table and an appropriate replacement is found there. For the *routing table*, one of the node belonging to the same row is asked for the entry it has at the same spot. Finally, for the *neighborhood set*, the node asks its neighbors for their neighborhood tables, checks the distance and picks the closest node.

Content Addressable Network

The Content Addressable Network structures the overlay network following a multi-dimensional cartesian coordinate space on a multi-torus. Each node is assigned a zone in this space according to a unique identifier that is hashed, usually its IP address. Once placed, the nodes construct their routing table that contains all the nodes that share a common edge along, at least, one of the dimensions of the virtual coordinate space. Each entry in the routing table contains the address and the coordinates of the zone controlled by the neighbor.

Like Chord[89] and Kademlia[57], CAN stores `{key, value}` pairs in the network. Whenever a node wants to store an object, it first computes a hash of that object to obtain its `key` and then associates a `value` that leads to its retrieval.

To route a query towards a destination each node on the path computes the distance between the zone of each of its neighbors and the destination, and forwards it to the closest one. This routing mechanism is used both for insertion and look up.

CAN achieves a routing performance of $O(d \times N^{1/d})$ — where d is the number of dimensions and N the number of nodes — and each node has to keep a routing table with, on average, $2 \times d$ entries.

To join the network a new node contacts a node already present (relying for example on bootstrap peers) and gives it a random point P. This point is looked up in the CAN network as if it were a look up query. Once at its destination, the node managing the zone containing P will hand over half of the zone to the new node, the keys belonging in that zone and the list of its neighbors.

Leaving the network results in the execution of a takeover algorithm by the neighbors of the departing node. The neighbor with the smallest region will inherit the zone, so as to keep all the zones roughly at the same size.

To increase the performances of the routing *CAN* proposes a number of enhancements: by permuting the dimensions of the coordinate space, $(x, y, z) \rightarrow (y, x, z)$, independent coordinate

spaces can be created where each node is assigned a zone in those spaces, called *realities*. Data are replicated in each of the realities, thus providing even more routing paths. For further data availability improvements, several hash functions could be used to insert data in as much locations of the virtual space. To reduce the latency between nodes, *landmarks* can be placed at strategic points in the virtual space such that new nodes first measure the round-trip-time they have with them, then sort the landmarks in ascending order, and finally inherit a zone associated to that ordering. For a more uniform partitioning of the virtual space, optimization strategies can be employed. For instance, instead of inheriting the zone pointed to by the (random) point P, a new node could first check if a larger zone exists in the vicinity and inherit half of that larger zone instead.

Kademlia

Kademlia[57] takes the same approach as CAN: nodes are assigned a 160-bit unique identifier using a hash of another value, `{key, value}` pairs are stored and a routing based on the identifiers of the nodes is used to efficiently locate a provided key.

Nodes in a Kademlia DHT are treated as leaves in a binary tree where the position of a node is determined by the shortest unique prefix of its identifier. Kademlia ensures that each node knows at least another node in any of the subtrees in which it is not included (see the circles in the figure below). For this purpose, whenever a node transmits a message it also includes its identifier, permitting the recipient to record it if necessary.

Leveraging this property, any node can find a key by successively querying the closest node·s known: node A asks the α closest nodes it knows to locate a key, which either return the possessor or the α closest nodes (α is a system-wide concurrency parameter). To obtain the distance between two keys, Kademlia computes the *exclusive or*, `XOR`, and interprets the result as an integer: $d(x, y) = x \oplus y$. Like CAN and Chord, this metric is symmetric but, unlike CAN, it is unidirectional which means that all lookups for a key will converge along the same path, no matter the origin.

Each node maintains a routing table composed of *k-buckets*. A *k-bucket* is a set of at-most k nodes, sorted according to the time of the last communication. The nodes contained in the i-th k-bucket are all at a distance comprised between 2^i and 2^{i+1} to the current node. The system parameter k is chosen so as to ensure that for any given set of k nodes, it is very unlikely that they will fail within an hour of each other.

When a node joins the network the most crucial part is for it to generate its k-buckets. It first contacts a node already presents, inserts that node in the appropriate bucket, then looks up its own key to make a first update, and finally updates, if necessary, the buckets located further away.

Chord

Chord[89] represents the network as a circle where each node is attributed a unique position given by its identifier. To generate identifiers Chord uses a variant of *consistent hashing* to uniformly distribute the load among the nodes as they are assigned roughly the same amount of keys. Like CAN and Kademlia, `{key, value}` pairs are stored, the routing is based on the identifiers, every look-up is resolved in $O(\log_2(N))$ messages (where N is the number of nodes) and requires information about only a fraction of the nodes to be performed.

To join the network, a node hashes its IP address to obtain its identifier and thus deduce

its position on the circle. The key k of an object is assigned to the first node which identifier equals or succeeds it in the identifier space. In that scenario, the node is called the *successor node* of the key k.

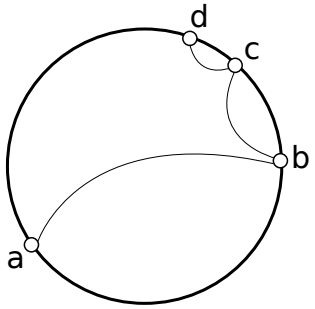

Figure 2.1: Routing from node a to d in a Chord network

To efficiently route look-up queries in the network, nodes in Chord maintain a *finger table*. The i^{th} entry in the *finger table* contains the identity of the first node that succeeds the current node by at least 2^{i-1} on the identifier circle. If the identifier circle ranges from 0 to 2^m then there are at most m entries in the *finger table*. By construction of that table, each nodes knows more about its direct vicinity than about nodes farther away. This property ensures that once a request is near its destination it will be correctly routed.

As nodes come and go, each node remaining in the network must be certain that its *finger table* is up to date. To do so, Chord uses a *stabilization* protocol: all nodes periodically run a `stab` procedure to learn of recent changes. This procedure consists in asking its successor for its predecessor: node n asks node s for its predecessor p, if $p \neq n$ and p is closer to n than s (s could have an outdated predecessor value) then n updates its successor to p.
To further negate the impact of node failures, each node maintains an additional list of successors, containing the first r successors. The value of r is chosen such that the simultaneous failure of the r successors is extremely unlikely.

To sum up, the different flavors of Distributed Hash Tables all offer the same features: a scalable, efficient, fault tolerant, and adaptable solution to the problem of organizing a fully-distributed network of nodes and of storing and retrieving information.

Although possessing the desired properties, Distributed Hash Table were not built with security nor privacy in mind: they lack protections when faced with malicious nodes. We thus discuss next the most important threats they can face as well as possible countermeasures.

2.3.3 Considered Attacks on Peer-to-Peer Systems

Distributed Hash Tables have shown to be relatively difficult to protect against security attacks. In the following, we present the three most important attacks that malicious nodes belonging to the network can launch as well as the main countermeasures for each: the *Sybil attack*, the *Eclipse attack* (or *Routing table poisoning*), and *Routing and Storage attacks*. Before concluding this section, and even though it is not a direct attack against DHT, we finish by discussing an issue related to our topic: how to reach consensus in a "Byzantine" setting.

Sybil Attack

When joining a network comprised of unknown nodes the first problem that one is faced with is that of *trustworthiness* — especially more so if you are supposed to exchange personal data with them. In our specific case, we partially solve this issue by associating each node with a hardware-based Trusted Execution Environment, meaning that as long as the sensitive manipulations are done by that unit the node is *trustworthy*[5]. However, how can we be sure that this node possesses such device? And if we can be sure that a node possesses a TEE, can we be sure that another node (or more) that also possesses one is not, in fact, the same? This attack, forging several identities, is called a *Sybil Attack*[29].

To perform a Sybil attack, a malicious party creates "bogus" nodes and makes them interact with the rest of the network as if they were distinct peers. Once implanted, they can disturb the normal flow of events: in our case, they could log all transmissions going through them (DHT lookup or storage operations), isolate and monitor specific nodes, or drastically increase the probability of being selected as actors in a distributed computations (by flooding the network).

To cope with this threat a globally accepted strategy[29, 99] is to rely on a central trusted identification authority. This authority produces for each node a cryptographic signature of a distinctive element it possesses. Paired with the public key of the authority, this signature is sufficient proof to attest the identity and the uniqueness of the node — as long as the authority is not corrupted. The ICANN is an example of such authority for the Domain Name System. The authority can either be online or offline, however an online authority constitutes an ideal target for attackers. Moreover, this hypothesis contradicts our targeted system as it would be the equivalent of a supporting server.

Other renowned strategies only partially eliminate *sybils*, only tackle a subpart of the problem (provide a correct routing despite *sybils*) or do not offer as strong guarantees as a central authority[72]. To do so they either make use of a social network providing trusted relationships between nodes like Whanau[47], Persea[3] or X-Vine[63]; test the resources the peer has at its disposal[86, 29]; or use redundancy mechanisms[26].

Preventing Sybil attacks is an important step to accomplish in a fully-distributed system as this attack opens up the way for more.

Eclipse Attack

To efficiently route queries, nodes maintain links to other peers — commonly referred to as neighbors — creating a routing table. The idea behind an Eclipse attack is to pollute the routing table of honest nodes in order for malicious nodes to "eclipse" other honest nodes in the network, i.e. erase them from the routing table of their neighbors. This attack is also known in the literature as `routing table poisoning`.

Networks that do not have special verifiable requirements for their neighbors are the most susceptible to this: whenever an honest node is updating its routing table, malicious nodes can falsify their information so as to insert themselves in the honest node's routing table. Another attack scenario is to place malicious nodes close to each other and from there on they can collude and attack neighboring honest nodes. Note that this attack does not necessarily have to be widespread but can be localized so as to isolate few selected nodes.

Similar to the Sybil attack, the most basic and most effective line of defense is to provide

[5]Strictly speaking, it is as trustworthy as the TEE.

nodes with random and stable identifiers[99]. Relying on a central authority to issue them is the preferred and most robust approach.

The main downside of this strategy is that it prevents performance optimization: the nodes present in the routing table are not selected because of their close geographical proximity. The majority of the literature about Eclipse attacks focuses on how to preserve these optimizations, unfortunately none of these solutions guarantee proper operation of the DHT unless they are associated with other mechanisms, which introduce a trade-off between performance and complexity[99].

Now that an attacker cannot flood the network with false nodes or poison the routing table of honest nodes, its remaining strategy is to provide false answers to query lookups.

Routing and Storage Attacks

Routing and storage attacks directly disrupt the execution of a DHT: once a malicious node receives a lookup request it can forward it to a non-existing, incorrect or malicious node, pretend to be the node responsible, claim that the object does not exist or answer with fake data.

The most common defenses are based on two mechanisms[99]: redundant storage and redundant routing.

Achieving redundant storage is done by replicating data at several places in the DHT: at nodes that are numerically close in the identifier space[89, 18], or at locations spread over the identifier space[105]. Having close replicas ease the maintenance of these replicas but it requires the malicious nodes to be spread over the identifier space. This means that nodes cannot control where they are inserted in the identifier space, once again prompting for a random and stable identifier.

Redundant routing can be implemented using *multiple paths*[18], *wide paths*[101] or *multiple wide paths*[11]. Multiple paths routing consists in sending copies of the lookup message to either the different replicas or to the initiator's neighbors if the replicas are close to each other. Wide paths routing returns at each step of the routing the k closest neighbors instead of the closest one, hence only requiring one honest node at each step. Multiple wide paths combine both approaches, trying wide paths successively.

In all these strategies having the malicious nodes evenly spread over the identifier space ensures a better probability of successful routing: they cannot isolate a node or a key if they cannot control their location to block all possible routes.

In addition to these mechanisms, some protocols[102, 101] also use an active central authority: Myrmic[101] makes the central trusted authority sign the routing table of the nodes, while Octopus[102] makes honest nodes actively detect their malicious counterpart and uses a certificate authority to issue and revoke the nodes' certificates.

Hence, having an offline central authority provide random and verifiable identifier is the best approach to successfully prevent an attacker from disrupting a DHT.

To finish up this section, and even though it is not a direct attack against distributed systems, we lastly want to discuss how to achieve consensus in this setting.

Byzantine Fault Tolerance

Agreeing on a common value or on a set of actions becomes mandatory whenever we have to answer some of the questions: which node·s take·s care of which action·s? In which order?

For which computation?

Byzantine fault tolerance (BFT) deals exactly with this: how to reach consensus in a distributed system in spite of a certain number of "faulty nodes" — that have either truly failed or are malicious. *BFT* was first formally studied in [71] and later introduced as the "Byzantine Generals Problem" in [46]. The conclusions of both are identical and expose two scenarios: (i) if the messages are not signed, can be forged but the absence of one can be detected (i.e. messages are time-bounded) then this problem finds no solution if one-third or more of the nodes are "faulty", (ii) if the messages are signed and cannot be forged, the problem becomes simpler and consensus can be reached for any number of attackers.

In the first case, a possible solution consists in considering a *majority with a fail-safe*: nodes will keep the value that has the most votes or, if no value appears more than the others, a pre-established fail-safe.

In the second case, the decision is left to a function CHOICE (computed by all nodes) that either returns all the valid values obtained or, if no value was provided, a pre-established fail-safe.

M. Castro and B. Liskov proposed *PBFT*[17] an algorithm based on state machine replication that overcomes the time-constrained and inefficient solutions proposed in earlier works — meaning that it can function on an asynchronous network such as the Internet — at the cost of not being able to solve the consensus problem. Indeed, it focuses on providing the core functionalities of an online information service: as long as less than $\lfloor \frac{n-1}{3} \rfloor$ nodes are faulty, *PBFT* ensures "safety" and "liveness". That is, operations are executed atomically and sequentially, and clients eventually receive an answer to their requests. *PBFT* roughly works as following:

1. A client asks a *primary* node a request;
2. the *primary* forwards it to all the *replicas*;
3. they all execute and send their answer to the client;
4. the client waits for $f + 1$ identical answers before accepting it as the result.

More recent works build on and/or improve *PBFT*: *Q/U*[1] uses quorum-based protocols to reduce the number of nodes involved in the processing of a query but with an increased overall cost, *Zyzzyva*[42] leaves to the client a part of the process to alleviate the charge on the servers, *Aardvark*[22] creates a "Robust-BFT" and uses game-theory strategies to anticipate failures and optimize for those worst-case scenarios, and *Abstract*[9] proposes an abstraction that can be used to help design resilient systems that are comprised of ad-hoc components — *Abstract* instances — used to handle specific conditions, hence not limited to a unique "flavor" of *PBFT*.

As discussed in the next chapter, despite their benefits, these solutions are not sufficient: even if it is possible to reach consensus and agree on a value if certain conditions are met, they cannot guarantee nor prove that this value was randomly generated. This last property is indeed a crucial component in the architecture we propose. Moreover, in our context, our proposed solution is much more efficient as it involves very few nodes.

2.3.4 Conclusion

In this section we saw two ways of organizing a fully-distributed network of nodes: using an unstructured or a structured peer-to-peer system.

Unstructured peer-to-peer systems have the advantage of requiring the nodes to store and compute very little information: they mainly have to forward requests. Operating this way is however inefficient and can even be extremely costly: broadcasts, flooding, or random-walks strategies are used to route queries within the network.

Structured peer-to-peer systems prefer the opposite strategy: by putting slightly more strain on the nodes and making them maintain routing tables and neighbors' lists, they can efficiently route the queries. Indeed, for most considered DHT an average of $\log_2(N)$ (N being the total number of nodes) messages is needed in order for a lookup query to reach its destination. Thus, to organize our network of home-cloud plugs augmented with tamper-resistant hardware we choose to rely on a Distributed Hash Table. Considering that no variation of the DHT emphasizes on protecting the privacy of the users we do not advocate against any and remain agnostic on the variation of DHT to use in our system.

We lastly saw the typical threats faced by a DHT system: Sybil attacks, Routing table poisoning, as well as Routing and Storage attacks. The main teaching this study provided is that to properly prevent these attacks, a central offline authority that delivers certificates offers the best defense mechanisms. These certificates first prevent malicious nodes from spawning fake nodes as they cannot counterfeit the certificates. Second, we can leverage these certificates to place the nodes on the overlay network and ensure a random and uniform distribution of the malicious nodes. As they cannot control their location on the overlay, their leeway to disrupt the DHT is severely hindered.

Chapter 3

Secure Actor Selection

Contents

In the related work we depicted the landscape in which this thesis takes place: we provide each user with a PDMS that contains most of their digital life, we secure them thanks to a *Trusted Execution Environment* and we finally organize them in *Distributed Hash Table* to ease the discovery of other peers while keeping a lightweight structure (i.e. the DHT does not need impose a heavy toll). As we also saw, no current framework allows the creation of a scalable and privacy-preserving distributed query system: cryptography-based solutions do not scale, non-interactive privacy-preserving data publishing technologies do not offer sufficient protection. Then, in this context, how can we build a scalable and privacy-preserving distributed query system?

This chapter lays the foundations of the solution we propose. To do so, we start by analyzing in details our problematic: what are we trying to achieve? What do we protect? Against whom? What are the requirements that a solution we deem acceptable should meet? How can we implement them? To answer these questions, we have to precisely define the data

we manipulate and the threats we want to defend against. In turn, formalizing those aspects leads to two subproblems: (i) randomly choosing PDMS nodes to act as actors in the query and (ii) assigning specific roles to ensure the "safe" execution of the query. Indeed, in a fully distributed system, relying on a single node would be contradictory meaning that we have to distribute the computation. However, an inappropriate distribution of the tasks and/or of the assignments would be equally disastrous: we have to assign both carefully. Formalizing these elements paves the way for our core design principles: *knowledge dispersion*, *task atomicity*, *hidden communications* and *imposed randomness*.

We follow these explanations, and conclude this chapter, with our first contribution: *SEP2P*[55] — a Secure and Efficient Peer-to-Peer protocol to randomly select nodes. This protocol leverages properties of the DHT (uniform distribution of the nodes, measuring the distance between them) to propose an algorithm that selects nodes in a distributed system in a way that is, at the same time, secure, random and efficient. The security and randomness stem from the fact that we know, with a high probability, that at least one honest node contributed to the creation and attestation of this list of nodes; while the efficiency stems from the fact that very few nodes are involved in this process.

3.1 Architectural design: objectives and threat model

3.1.1 Base System Architecture

As we evolve in a peer-to-peer system and only rely on the PDMS nodes, this means that each node may play several roles:

Node role 1: *Each node is a potential* ***data source****. For instance, producing sensed geo-localized data about the local traffic speed, or sharing grades used to compute recommendations.*

Node role 2: *Given the fully-decentralized nature of our system, each node is a potential* ***data processor****, also called* ***actor****, providing part of the required processing.*

Node role 3: *The initiator of a distributed processing is called the* ***Querier (Q)****. Q could be any node with participatory sensing applications, or the query issuer in distributed query or data diffusion applications.*

Relying on a fully-distributed system induces several problems, e.g., integrating new nodes, maintaining a coherent global state, making nodes that do not know each other interact, handling churn, maintaining some metadata. It thus requires a communication overlay allowing for efficient node discovery, data indexing and search. Fortunately, as stated in the previous chapter, these problems have already been extensively studied in the literature and the *Distributed Hash Tables* (DHTs) appear to be the solution reaching consensus. Hence, we leverage classical DHT techniques as a basis for communication efficiency and scalability.

3.1.2 Security Considerations

We use the terminology of ARM[93] to designate the three attack levels on a PDMS node, i.e., *hack*, *shack* and *lab attacks*. A *hack attack* is a software attack in which the attacker (the PDMS owner or remote attacker) downloads code on the device to control it. A *shack attack* is a low-budget hardware attack, i.e., using basic equipment and knowledge. Finally,

a *lab attack* is the most advanced, comprehensive and invasive hardware attack for which the attacker has access to laboratory equipment, can perform reverse engineering of a device and monitor analog signals. Note that shack and lab attacks require a physical access to the device and that TEEs are designed to at least resist hack and shack attacks.

Our threat model considers four security assumptions:

Assumption 1: *Each PDMS is locally secured by using TEE-like technology flourishing nowadays (e.g., [36, 45, 74]).*

This assumption is reasonable considering that a PDMS is supposed to store the entire digital life of its owner. A major security feature of TEE technology is to provide isolation, i.e., strong guarantees that the local computation inside the TEE cannot be spied upon, even in the presence of an untrusted computational environment. Hence, to break to confidentiality barrier of a TEE, a lab attack is mandatory. This has an important consequence: *an attacker cannot conduct a successful attack on a remote node, i.e., not under her possession.*

Assumption 2: *Each PDMS device is supplied with a trustworthy certificate attesting that it is a genuine PDMS.*

Without this assumption, an attacker can easily emulate nodes in the network, and conduct a Sybil attack[18], mastering a large proportion of nodes (e.g., playing the role of data processor nodes), thus defeating any countermeasure. Note that this does not require an online PKI: the certificate can be attached to the hardware device and not to the device owner.

Assumption 3: *Corrupted nodes by a lab attack behave like covert adversaries.*

In other words, they derive from the protocol to obtain private information only if they cannot be detected[10], as detected malicious behavior leads to an exclusion from the system. This exclusion can be enforced by fairly efficient and scalable revocation mechanisms such as: Merkle Hash Tree based certificate revocation[66], efficient distribution of revocation information over P2P networks[50], or scalable PKI based on P2P systems[65].

Assumption 4: *The communications of a node can be spied upon by either its owner or a remote attacker.*

Considering this kind of attacks is motivated by two main reasons. First, network port spying does not require a lab attack but merely a hack or shack attack which makes it (i) easier and thus more likely and (ii) doable locally and remotely. Second, in a purely distributed system, even though node communications are ubiquitous, they can reveal private information to an attacker: either through the *data* itself — sensitive information are sent unencrypted and are de facto subject to interception; or through the *metadata* — two specific nodes are exchanging messages at a specific time.

3.1.3 Threat Model

The above considered assumptions already offer a certain level of security at the node and system levels. Yet, no hardware security can be described as unbreakable. Therefore, our threat model considers that an attacker (e.g., one or several colluding malicious users) can possess several PDMSs and conduct lab attacks on these devices, thus mastering several corrupted nodes which can collude. For simplicity, we call them *colluding nodes*.

It is important to notice that the worst-case attack is represented by the maximum number of colluding nodes in the system (i.e., controlled by a single entity). Corrupting few nodes

can lead to some private data disclosure, but this disclosure is very limited in a well-designed system with a large number of nodes. Therefore, an attacker needs to increase the collusion range to fully benefit from the attack (i.e., access a significant amount of private data).

Thereby, the remaining question is: how many colluding nodes could an attacker control in the system? The main difficulty for an attacker is that colluding nodes must remain indistinguishable from honest nodes (see Assumption 3). Since PDMSs are associated to "real" individuals (e.g., by delivering the device only to real users proving their identity), collusions between individuals remains possible (hidden groups) but such collusions cannot scale without being minimally advertised, hence making them distinguishable and breaking their cover. Thus, wide collusions are extremely difficult to build since it requires significant organization between a very large number of users, which in practice requires an extremely powerful attacker as well as extreme discretion, and are thus the equivalent of a state-size attack. Although worrisome, a situation with such a large proportion of colluding nodes does not prevent our system from functioning and from completing our objective: as we show in the experiments (see Section 5.2.1) we satisfy the requirements while maintaining an average leak that is under-linear with regard to the maximum number of colluding nodes controlled by an attacker — *no matter that number*. However, do note that considering a large proportion of colluding nodes (e.g., 10%) in a fully-distributed system is vain as it would inexorably lead to large disclosure whatever the protocol having a reasonable overhead (e.g., outside the MPC scope). Hence, we consider that a very powerful attacker could control up to a small percentage (e.g., up to 1%) of the nodes, which corresponds to a wide collusion requiring a lab attack on these nodes as well as a highly organized collusion between the owners of the nodes.

What does the system protect? Our objective is to offer the maximum possible confidentiality protection of the user private data under the above considered threat model. Many other issues related to statistical databases (e.g., inferences from results, determining the authorized queries, query replay, fake data injection, etc.) or to network security (e.g., message drop/delay, routing table poisoning[99]) are complementary to this work and fall outside of the scope of this paper.

Confidentiality with partial integrity? Although TEEs have the means to attest that the intended computations were performed by the different nodes[44, 43], we cannot hope to maintain those capabilities when the TEE is fully compromised. Thus, in this conditions, we do not claim to offer integrity. Therefore, assuming that a proposed computational plan is guaranteed to offer confidentiality, it is debatable whether this confidentiality can still be achieved when the computation is altered. Assessing the loss of confidentiality is no easy task as there are numerous cases to consider: the type of data manipulated, the computations, the distribution of the data, the envisioned collusions. Note that in our context, not every and all aspects of the execution can be modified, the leeway an attacker has is limited to the addition of bogus results or the inclusion of more nodes in the query execution process. This restrained scope pleads in favor of a reduced impact but this study, mainly due to its sheer size, falls outside of the scope of this work.

3.1.4 Requirements

Given the considered threat model, we derive in this section the requirements that any solution, in this specific context, must address to protect the data privacy of the users. Since we cannot exclude having colluding nodes in the system and since the colluding nodes behave like covert

adversaries, private information leakage is unavoidable. Under these conditions, the best countermeasures one can take are: (i) *minimize the risk* of a data leakage, i.e., reduce at most the probability of a leakage to happen; and (ii) *minimize the impact* of a data leakage, i.e., reduce at most the leakage size. Obviously, these countermeasures should not generate overheads that render the system unpractical. This leads to:

Requirement 1: (security) Random actor selection. Ensure that colluding nodes cannot influence the selection of the data processor nodes.

As evident as this may be, it is crucial that colluding nodes cannot favor themselves in the attribution of the roles: even if those roles are extremely well thought and disclose the absolute minimum amount of information, not preventing this behavior changes the probability of a leakage for any and all queries from "unlikely" to "absolute certainty".

Requirement 2: (security) Knowledge dispersion. No single node (or few nodes) should store a significant amount of sensitive data, unless it owns these data.

The rationale behind this requirement is straightforward: to not create a "central" point in the network as it would drastically increase the benefits of compromising this particular node (compared to compromising a large portion of the network).

Requirement 3: (security) Task atomicity. Data tasks should be atomic, i.e., reduced to a maximum such that it minimizes the required sensitive data to execute the task.

This requirement is similar to the principle of *compartmentalization* in information security, which consists in limiting the information access to the minimum amount allowing an entity to execute a certain task. Typically, a node can execute a subtask without knowing the purpose or the scope of the global task. Dividing a given distributed computation in atomic tasks obviously depends on the precise definition of that computation. Hence, we restrict our analysis in to sketches of solutions for the three application classes considered in this paper.

Requirement 4: (security) Hidden communications. Sensitive data and metadata should be protected such that no attacker can gain knowledge by listening to the communications of a limited set of nodes.

In our context "hidden" can either mean *encryption*, i.e. relying on cryptography to protect the content of the exchanged messages, or *anonymous communications*, i.e. hiding the origin or the destination of a communication to an attacker spying on the network. Saying that an attacker spies on a "limited" set of nodes excludes state-size attacks where the communications of *all* the network are spied upon. As we consider network that can possibly house hundreds of thousands or millions of nodes, spying on all of them would require tremendous resources.

Requirement 5: (efficiency) Security overheads. Minimize the number of costly operations, e.g., cryptographic signature verifications or communication overhead, and ensure system scalability with an increasing number of nodes or colluding nodes.

The need for efficiency stems from two fundamental prerequisites for mass adoption: obtaining a result in a matter of a few moments and making sure that PDMSs are not overwhelmed by the distributed computation. Giving the end-user the result in a short span makes for a viable user experience: if the results come "late" (although this notion is purely subjective) then no user will use this system. Not overwhelming the PDMSs means that they can perform their primary functionality unhindered, once again making for a viable user experience: if, because of the distributed computations, the users cannot properly benefit from their PDMS, then they might block distributed computations.

Independently of the distributed protocol chosen to implement some given application, the system must delegate the data-oriented tasks to randomly selected nodes, in accordance with our first requirement. Therefore, the random selection protocol is generic and constitutes the security basis of any distributed protocol in our system. However, given the considered threat model, it is challenging to design an actor selection protocol that is both secure and efficient. In the rest of this chapter we detail SEP2P, our first contribution, which specifically addresses this problem.

3.2 SEP2P: Secure Actor Selection

Let us first detail some useful classical cryptographic tools focusing on the properties used in our protocol.

A **cryptographic hash function**[60] is a one-way function that maps a data of arbitrary size to a fixed size bit string (e.g., 224 bits) and is resistant to collision. An interesting property of hash functions is that output distribution is uniform. In the following, $hash()$ refers to a cryptographic hash.

A **cryptographic signature**[60] can be used by a node n to prove that a data d was produced by n (authentication) and has not been altered (integrity). The signature is produced by encrypting $hash(d)$ using the private key of n. Any node can verify the signature by decrypting it using the public key of n and comparing the result with $hash(d)$. The signature includes the signer public key certificate, $cert_n$ (see Assumption 2).

We consider a system of N nodes, in which we want to randomly select A actors, despite wide collusion attacks from C colluding nodes. The main notations are summarized in Table 3.1.

3.2.1 Effectiveness, Cost and Optimal Bounds

Ideally, we would want to ensure that all A actors are honest, but this is impossible, since colluding nodes are indistinguishable from honest nodes. Therefore, the best achievable protection is obtained when actors are randomly selected and the selection cannot be influenced

Notation	Explanation
N	Total number of nodes in the system
A	Number of actor nodes (data processors)
C	Maximum number of colluding nodes ($C \geq 1$)
A_C	Average number of corrupted actors for a given protocol
A_{ideal_C}	Average number of corrupted actors for an ideal protocol
Q	Querier (starting the execution)
k	Security degree
α	Security threshold
S	Execution Setter node, computing the actor list
R_i, rs_i	DHT region R_i of size rs_i

Table 3.1: Main notations for SEP2P

by C colluding nodes, i.e., the average number of corrupted selected actors in the ideal case is $A_{ideal_C} = A \times C/N$ ($A_{ideal_C} > 0$). Thus, the impact of a collusion attack remains proportional with the number of colluding nodes, which is the best situation given our context. This guarantees that the attacker cannot obtain more private information than what she can passively get from observing the information randomly reaching its colluding nodes.

The following definitions quantify the *security effectiveness* and *security cost* of an actor selection protocol.

Definition 1. Security effectiveness. *The* security effectiveness *of an actor selection protocol is defined as the ratio between A_{ideal_C} and the average number of corrupted selected actors for the measured protocol (A_C), i.e., security effectiveness $= A_{ideal_C}/A_C$. The security effectiveness has maximum value (i.e., 1) when $A_C = A_{ideal_C}$ and minimum value (i.e. C/N) when all the actors are corrupted.*

Definition 2. Verifier node. *A* verifier node *is a node who needs verifying the actor list before delivering sensitive data, e.g., a data source.*

Definition 3. Security cost. *The* security cost *of an actor selection protocol is defined as the number of asymmetric cryptographic operations, e.g., signature verification, required by verifier nodes to check the selected actor list.*

Note that the security cost considers only the verification of the actor list and not the cost of building the list. The rationale is that the verification cost has a larger impact on the overall performance since the number of verifier nodes can be high in a large distributed system: data sources need to verify the actor list before delivering their data. Other performance related issues (cost of the actor list generation, load balancing, maintenance costs) are discussed in Section 3.3.3.

Optimal bounds. The best possible case one could expect in terms of security effectiveness and cost in our context can be achieved using an idealized trusted server that knows all the nodes and provides a different random actor list for each system computation. This ideal solution reaches a maximal security effectiveness and a security cost of 1, since any verifier node must only check the signature of the trusted entity.

Evidently, this solution in not acceptable since it represents a highly desirable target for attackers, i.e., a central point of attack and contradicts the fully distributed nature of SEP2P. Therefore, we need distributed solutions relying only on the nodes. To underline the existing tension between security effectiveness and cost, we discuss two basic distributed protocols for the actor selection, focusing either on the security cost or on the security effectiveness. To simplify the protocols description, we initially assume a full mesh network overlay, i.e., each node knows the complete list of nodes in the system and its evolution over time.

Baseline cost-optimal protocol. The Querier (Q) selects randomly the actors. The security effectiveness is minimal: $A_C = \min(A, C)$ since Q may be corrupted (which is the case when any node can trigger a computation). There is thus no necessity to provide any signature: the security cost is 0.

Baseline security-optimal protocol. Proposing an optimal protocol in terms of security is challenging in a decentralized architecture (without any supporting trusted party) and considering covert adversaries. This conjunction leads to a situation where no single node in the system can claim to securely provide a list of actors (the provider itself can be corrupted). The work in [12] proposes the CSAR protocol which provides a secure way to generate a verifiable random value under the condition that there is at least one honest node participating in the distributed protocol — a more detailed explanation is given below. Applying to our context, we can ensure generating a real random value only if there are at least $C + 1$ participating nodes. Also, once we obtain a verifiable random value, we can derive up to A random values by repeatedly hashing the initial value $A - 1$ times. The final step is to map the set of A random values to the nodes. This can be easily done, e.g., by sorting the nodes on their public key and associating the random value to a rank in the sorted list. This protocol has an optimal security effectiveness, i.e., 1, since the actors are guaranteed to be selected randomly. On the other hand, checking the CSAR results requires one signature verification per participant. Thus, the security cost is $C + 1$ asymmetric cryptographic operations per verifier node. Since C can be large, such a solution cannot scale with large systems and wide collusion attackers as it would lead to an extreme verification cost.

CSAR is a technique for generating ***Cryptographically Strong, Accountable Randomness***. It proposes a pseudo-random generator that (1) outputs cryptographically strong randomness, (2) is accountable, i.e. after each random value r is generated, it should be possible to generate a proof that this value was correctly derived from a given seed, (3) ensures that future random values are unpredictable, and (4) ensures that the previous properties hold even if malicious nodes are present while the seed is computed (and that, in particular, no node should be able to influence the output). Additionally, both the generation of the randomness and its verification are efficient, in order to limit the cost of accountability.

CSAR achieves these goals with a two steps protocol: the first step is the *setup* where a seed an a permutation are generated, and the second is the actual *generation of random values*.

Our work only leverages the coin-toss subprotocol, part of the *setup* and responsible for the generation of the seed, that "can easily be shown to produce a random value s, provided that at least one entity is honest"[12]. Its process is:

Entities $P, P_1, \ldots, P_k$ choose random values $r, r_1, \ldots, r_k$. Then each entity P_i computes $c_i := hash(r_i)$ and produces a signature σ_i on c_i. Next, all (c_i, σ_i) are sent to P. P sets $c := hash(r)$, $h := (c, c_1, \sigma_1, \ldots, c_k, \sigma_k)$ and produces a signature σ on h. Then each P_i checks all signatures in h, produces a signature σ'_i on h and sends (r_i, σ'_i) to P. Finally, P checks all signatures σ'_i and sends $(r, r_1, \ldots, r_k)$ to $P_1, \ldots, P_k$. The outcome of the coin toss is $s := r \oplus r_1 \oplus \ldots \oplus r_k$.

To achieve these security bounds, both protocols require a full mesh network overlay which is also extremely costly to maintain in practice, especially for large networks. This contradicts the efficiency and scalability requirement. Using a DHT overlay instead of a full mesh solves the problem of communication efficiency/scalability. However, this impacts the optimal bounds of both protocols. For the first protocol, the security cost increases from 0 to up to A since a verifier node which does not "know" any of the actors has to verify their certificates to be sure that the actors are genuine PDMSs (to avoid Sybil attacks). Similarly, for the second protocol, the security cost increases to $2(C + 1) + A$ for the same reason, i.e., checking that participant and selected actors are genuine PDMSs. Even worse, the optimal security effectiveness can no longer be guaranteed since with a DHT, there is no secure way of associating the random values to the nodes unless using secure DHT techniques[99] with a large impact on performance.

3.2.2 Overview of the Proposed Solution

To address all these problems, we propose a protocol that reaches maximal security effectiveness at a security cost of $2k$. k is called the *security degree* and is very small. Also, our protocol builds directly on a classical, efficient DHT overlay without requiring any modifications. We describe some important features in SEP2P which make this possible and then sketch the protocol.

Imposed and uniform distribution of node location: the node ID, used when inserting a node in the DHT, is imposed in SEP2P, in a way that leads to a uniformly distributed node location in the DHT virtual space. Consequently, colluding nodes are also evenly distributed in the DHT, thus avoiding spatial clusters. We use extensively this property to drastically reduce the cost of security by taking localized decisions (see below), i.e., limited to the nodes located in "small" regions in the virtual space. Achieving imposed node location is easy, based on the public key of the certificate of each node. We compute a cryptographic hash of this key, which is, by construction, uniformly distributed, and use this hash for insertion in the DHT virtual space. The advantages of using the public key are (i) its uniqueness; and (ii) the node location can be checked with a single signature verification.

Probabilistic guarantees: Given the imposed, uniform node location which applies indistinctly to honest and colluding nodes, we can have probabilistic guarantees on the maximum number of colluding nodes in a DHT subspace of a given size, called *DHT region* hereafter. We can compute the probability of having **at least k colluding nodes** (see Section 3.2.3) and choose the DHT region size such that the probability is very close to 0. In our context, we want to have a probability smaller than α, the security threshold. The main idea is to set α so that the probability of having k colluding nodes in the same region becomes so low that we can consider that it "never happens", e.g., $\alpha = 10^{-6}$ (see Section 3.3.1). Such a guarantee is used in the protocol sketched below and then detailed in the following subsection.

Sketch of verifiable selection protocol of A actors (see Figure3.1)

(1) Run a distributed protocol inspired from CSAR[12] to generate a **verifiable random value**, i.e., proven to have been truly randomly generated by k nodes if at least one is honest (see Section 3.2.4). The k nodes are selected in a DHT region R_1, centered on the triggering node (Q), whose region size rs_1 is set such that we have probabilistic guarantees to "never" (probability $< \alpha$) have k or more colluding nodes, i.e., at least one of the k nodes is honest.

(2) Map the hash of that random value into coordinates to define a location p in the DHT virtual space and contact through the DHT the node, called **execution Setter** (S), managing this location.

(3) S then selects k nodes (the actor list builders) in a region R_2, centered on p, using probabilistic guarantees, such that we "never" have k or more colluding nodes. Given the uniform distribution of the node on the virtual space, we have $rs_2 = rs_1$.

(4) Each actor list builder then selects A nodes in a region R_3, centered on p, whose size rs_3 is such that R_3 includes at least A nodes with high probability (see Section 3.2.6 and 3.3.3 for rs_3 tuning).

(5) Run a **distributed verifiable selection protocol** in the spirit of [12] such that the k nodes selected in (3) can: (i) check the validity of the random value generated in (1); (ii) build the actor list securely; (iii) sign both the random value and the list of A actors. This step is detailed in Section 3.2.5.

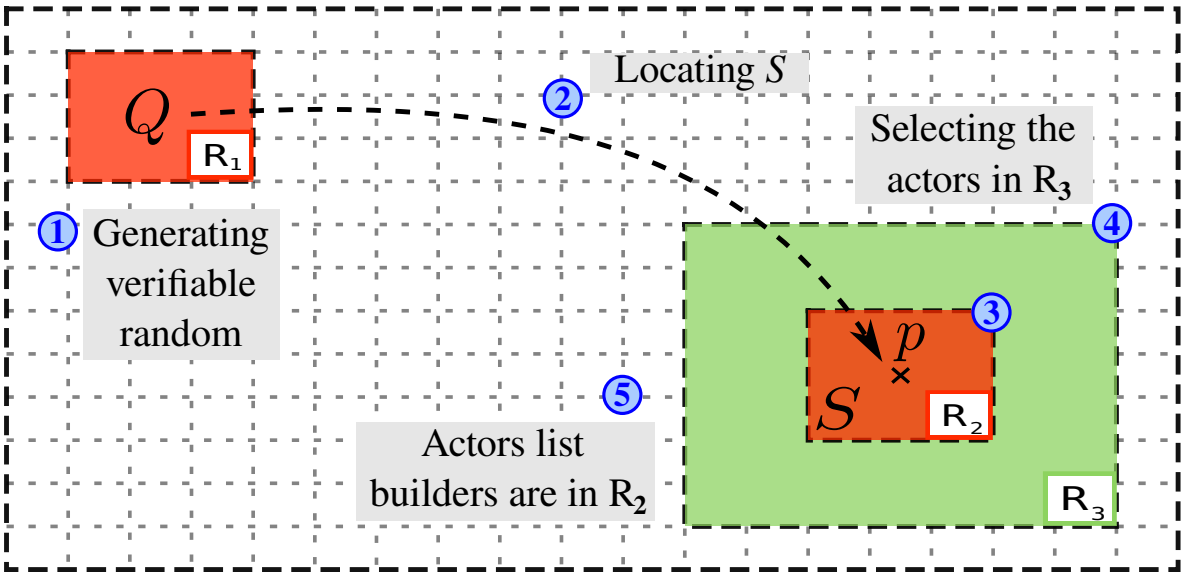

Figure 3.1: Sketch of verifiable selection

The result is a list of A actors that is signed by k nodes, among which at least one is honest. Doing so reduces the security cost to $2k$ asymmetric cryptographic operations: k to check the certificate of the k list builders, verifying that they belong in the region R_2, centered on p; and k to check each builder signature.

Notation	Explanation
$kpub_n$	Public key of node n
$cert_n$	Trustworthy certificate of node n
$sign_n$	Signature by node n (includes $cert_n$)
QL_i	execution Trigger Legitimate node i
RND_i	Random number generated by QL_i
$(V)RND_T$	(Verifiable) random generated by Q
SL_j	execution Setter Legitimate node j
RND_S	Random generated by S
CL_j	Partial candidate list of legitimate nodes w.r.t. R_3
CL	Candidate List of legitimate nodes
$(V)AL$	(Verifiable) Actor List

Table 3.2: Main notations for Sections 3.2.3 to 3.2.5

3.2.3 Providing Probabilistic Guarantees

To generate verifiable random values or validate the query actor selection, SEP2P employs distributed computations between a small subset of the nodes thanks to the notion of node legitimacy and probabilistic guarantees defined below using the notations in Table 3.2.

Definition 4. LEGITIMATE NODES. *Given a region R in the virtual space of a DHT, for any node i we say that node i is legitimate w.r.t. R if and only if $hash(kpub_i) \in R$.*

To be able to provide probabilistic guarantees as explained in Section 3.2.2, we need to estimate the number of nodes in a region:

Lemma 1. Let R be a DHT region of size rs in a virtual space of a DHT of total size 1 (i.e., normalized) and let N be the total number of network nodes with a uniform distribution of the node location in the virtual space. The probability, PL, of having at least m legitimate nodes in R is:

$$PL(\geq_m, N, rs) = \sum_{i=m}^{N} \binom{N}{i} \cdot rs^i \cdot (1 - rs)^{N-i} \tag{3.1}$$

Proof (sketch): Let us consider a partition of the N nodes into two subsets containing i and $N-i$ nodes. Since the distribution of nodes is uniform in space, the probability of having the i nodes inside R and the $N-i$ nodes outside R is $rs^i \cdot (1-rs)^{N-i}$ and there are $\binom{N}{i}$ possible combinations of generating this node partitioning. The probability of having at least m nodes in R is equal to the probability of having exactly m nodes plus the probability of having exactly $m+1$ plus... the probability of having N, which leads to the equation in(3.1). □

Application to colluding nodes: Let $C < N$ be the maximum number of colluding nodes. We can apply formula 3.1 to compute the probability, PC of having at least k colluding

nodes in R:

$$PC(\geq_k, C, rs) = \sum_{i=k}^{C} \binom{C}{i} \cdot rs^i \cdot (1 - rs)^{C-i} \tag{3.2}$$

We can notice that this probability only depends on C. It does not depend on the region center since we have a uniform distribution of the nodes on the virtual space.

3.2.4 Verifiable Random Generation

Our goal is to generate a random value, using k nodes and to guarantee that none of the k nodes can choose the final computed random value (or any of its bits). Any node in the system should be able to check the validity of this random value (i.e., to have proofs that it has been correctly generated). This is possible as soon as at least one of the k nodes is honest, this guarantee being obtained thanks to equation (3.2) by choosing the adequate size for the DHT region R and by using k legitimate nodes with regard to R.

A node Q wanting to generate a verifiable random, selecting a region of size rs_1 with $PC(rs_1) < \alpha$ centered on itself, executes:

Verifiable random number generation protocol

---

(1) Q contacts any k legitimates nodes QL_i ($i \in [1, k]$) w.r.t. R_1.
(2) Each QL_i sends $hash(RND_i)$ to Q, where RND_i is a random number (on the same domain as the hash function, e.g., 224 bits) QL_i generates.
(3) Once Q has received the k hashes, it sends back the list L of hashes to the QL_i: $L = (hash(RND_i))_{i \in [1,k]}$
(4) Each QL_i checks that $hash(RND_i) \in L$, and, in the positive case, returns $sign_i(L)$ and RND_i.
(5) Q gathers the k messages and builds the verifiable random: $VRND_Q = (cert_Q, (sign_i(L), RND_i)_{i \in [1,k]})$.

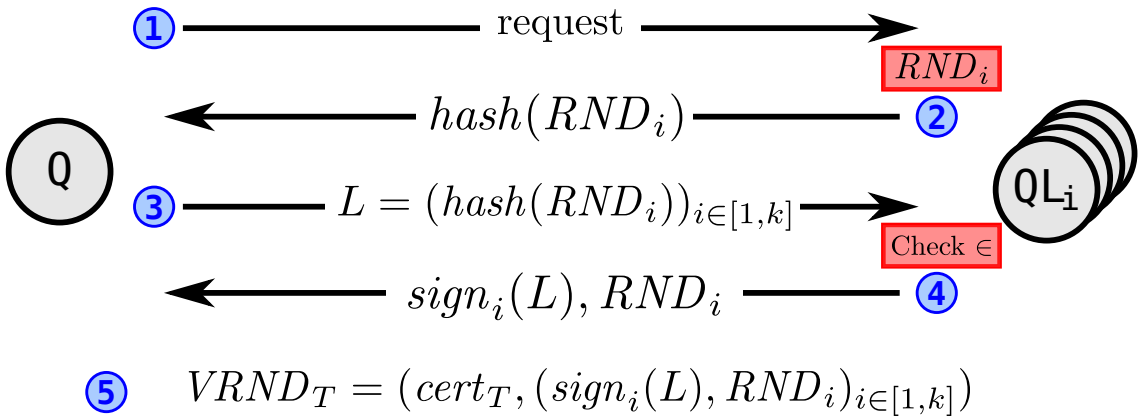

Figure 3.2: Verifiable random

The above random generation protocol is adapted from [12] which includes a formal proof. Note that the protocol in [12] does not include the notion of node legitimacy and thus needs

$C+1$ participating nodes instead of k. Intuitively, the nodes commit on their selected random value by sending its hash (Step (2)), and all the hash values are known by each of the k nodes before providing the final signature (Step (4)). Therefore, an attacker controlling $k-1$ QL_i nodes cannot influence the final random value since these nodes cannot change their random values (committed at Step (2)). Thus, the correct random value of a single honest node is enough to obtain a truly random final value RND_Q.

To obtain and check the verifiable random value, any node must: (i) check $cert_Q$ and compute L by hashing all RND_i; (ii) for $i \in [1,k]$, check $cert_i$, check the legitimacy of QL_i using $cert_Q$ and validate $sign_i(L)$. The final random value is:

$$RND_Q = RND_1 \oplus RND_2 \oplus \ldots \oplus RND_k$$

In (i), we verify that Q is a genuine PDMS, retrieve the center of the region R_1 and compute L, both being necessary for the next verification; (ii) starts by confirming that each QL_i is genuine, then it ensures that they are legitimate w.r.t the location of Q and R_1, after which it confirms the hash list by checking the signatures, and finally, it computes RND_Q.

3.2.5 Distributed Secure Selection Protocol

The main goal of the proposed protocol is to select the A actors such that this selection cannot be influenced by colluding nodes.

Definition 5. EXECUTION SETTER. *The **execution Setter** (S) is chosen randomly based on a verifiable random generated by Q. Its role is to coordinate the selection of the computation actors and to setup the execution by sending the appropriate information to each actor.*

In the following, we assume that each node n in SEP2P keeps a node cache, called $cache_n$, of the IP address and certificate of legitimate nodes w.r.t. a region of size rs_3 centered on node n location. The cache size and the cache maintenance cost are discussed in Section 3.2.6 and evaluated in 3.3.3.

SEP2P distributed secure actor selection protocol

(1) Q generates the verifiable random $VRND_Q$ (see Section 3.2.4).
(2) Q maps $hash(RND_Q)$ into coordinates and contact S through the DHT.
(3) S contacts any k legitimates nodes w.r.t. R_2, SL_j ($j \in [1,k]$) and sends to each $VRND_Q$ (see Section 3.2.4).
(4) Each SL_j sends $hash(RND_j \parallel CL_j)$ to S, where RND_j is a random number SL_j generates, and CL_j is the set of nodes from $cache_j$ which are legitimate w.r.t. R_3.
(5) Once S has received the k hashes, it sends back the list L_1 of hashes to all SL_j: $L_1 = (hash(RND_j \parallel CL_j))_{j\in[1,k]}$.
(6) Each SL_j checks that its own $hash(RND_j \parallel CL_j) \in L_1$ and, in the positive case, returns RND_j and CL_j.
(7) S gathers the k messages and sends to all SL_j the list $L_2 = ((RND_j, CL_j)_{j\in[1,k]})$.
(8) Each SL_j does the following:
  (a) Checks $VRND_Q$ and computes RND_Q (see Section 3.2.4).
  (b) Checks that each (RND_j, CL_j) from L_2 is consistent with the corresponding $hash(RND_j \mid CL_j)$ from L_1.

(c) Computes the union, after removing possible duplicates, of all CL_j to obtain a candidate list of legitimate nodes CL.
(d) Computes the $RND_S = RND_1 \oplus RND_2 \oplus \ldots \oplus RND_k$.
(e) Sorts CL on $kpub_n \oplus RND_S$ (where $kpub_n$ is the public key of a node $n \in CL$) and selects the A first candidates to build the actor list AL.
(f) Checks the legitimacy of AL nodes w.r.t. R_3.
(g) Signs (RND_Q, AL) and sends it to S.

(9) S gathers k results and builds the verifiable actor lists:
$VAL = (RND_Q, AL, (sign_j(RND_Q, AL)))_{j \in [1,k]}$

The goal of **steps (1) and (2)** is to displace the DHT region, where actors are selected, from Q to S with three benefits: (1) Q is likely to be corrupted (as any node is allowed to trigger a computation) while S is chosen randomly using the verifiable random protocol; (2) it distributes the potential leaks in a different region for each computation; (3) it balances the load on the whole SEP2P network thus improving the overall performance.

Steps (3) to (6) are similar to steps (1) to (3) of the verifiable random protocol, except that the signature by SL_j is delayed to step (8)g. Delaying the signature allows SL_js to check and attest the validity of $VRND_Q$ (step (8)a). The protocol cost is increased (since k nodes verify $VRND_Q$) but the verifying cost is reduced accordingly since having k SL_j signing RND_Q (step (8)g) means that it is correct (remind that at least one of the k SL_js is honest).

Steps (8)b to (8)e are dedicated to the actor list building (AL) based on the candidate list (CL) and deserve a more detailed explanation: in our context, in order to securely build the actor list, the k participants first have to agree on a common basis and then execute, in parallel, a procedure that is unpredictable and gives identical results to all participants. Since it is unpredictable we are certain that the inputs cannot be manipulated beforehand so as to influence the rest of the procedure. Since it gives identical results for all actor list builders, and since at least one node is honest, we are sure that no colluding node can alter the results. By sorting the nodes in CL using a verifiable random number and the public keys of the nodes fulfills both requirements: the random number takes care of the unpredictability, while the commitment of each SL_j on their intermediary lists in step (4), coupled with the XOR operation on the public keys of CL nodes, is a simple yet effective way of producing identical results.

In **steps (8)f and (8)g**, k SL_js check the validity of the result, i.e., that any actor of AL belongs to R_3 and attest it by signing the results. Note that this check is not necessary for any actor n in AL that was found in k CL_j since this fact attests that at least one honest node possesses n in its $cache_j$. Assuming $cache_j$ contains only genuine nodes (we say that $cache_j$ is valid — see Section 3.2.6) and since $rs_3 > rs_2$, most of the actors in AL are found in k CL_j, thus diminishing drastically the actor list building cost. Actually the validity of $cache_j$ is necessary to ensure that a colluding node selected as SL cannot hide honest nodes with the hope of having a larger proportion of colluding nodes in AL. Indeed, at least one of the SL is honest and provides its full $cache_j$ that is thus included in CL. We can observe that $cache_j$ can be actually seen as the relevant part (for node j) of a full mesh network, which offers its benefits without paying the whole maintenance cost.

To check the verifiable actor list (VAL), any verifier node must do: for $j \in [1, k]$, check $cert_j$, check the legitimacy of SL_j using RND_Q and validate $sign_j(AL)$. Thus, the verifying cost is limited to k certificate verifications and k signature verifications, i.e., $2k$ asymmetric crypto-operations. We show in Section 3.3 that k is generally lower than 6.

3.2.6 Protocol Implementation Details

In this section we discuss a few important implementation issues of the proposed actor selection protocol.

Sparse DHT regions: Despite the uniform distribution of nodes on the DHT virtual space, there is no absolute guarantee of not having sparse DHT regions. This can have two negative impacts on the SEP2P protocol: during the selection of k QL in R_1 (or k SL in R_2) and of the A actors in R_3. Both cases exhibit interesting trade-offs:

Choosing R_1 (or R_2) region size: on the one hand, a small rs leads to a smaller k value, which in turn reduces the protocol security cost. On the other hand, setting rs too small can lead to situations in which nodes have less than k legitimate nodes in their R region and as such cannot participate in the actor selection protocol (as querier or execution setter) which is problematic. For this reason, in SEP2P we provide a table of couples (k_i, rs_i), named **k-table**, which allows any node to find k_i legitimate nodes in the region of associated rs_i size. The k-table is computed thanks to PL and PC (equations 3.1 and 3.2) to ensure that whatever the couple chosen, the probability of having k or more colluding nodes remains equal. The largest k of the k-table corresponds to the region size allowing any node to find those legitimate nodes with a very high probability, i.e., 1-α, while lower values allow to reduce the security cost in denser network regions. Thus, the k-table optimizes the overall cost of the SEP2P protocol and warrants that any node can be selected as triggering node or execution setter.

Choosing R_3 region size: Choosing a too small rs_3 has a negative impact on the system performance. If the SL cannot find enough nodes in R_3, they can attest it (e.g., in step (8)c in SEP2P protocol) and S can use the k signatures to displace the actor selection to another region (e.g., selected by rehashing the initial RND_Q). This mechanism allows the protocol to be executed successfully even if some network regions are sparser. However, there are two drawbacks. First, the cost of the actor selection increases since (part of) SEP2P protocol must be executed twice (or more times). Second, this also introduces an unbalance in the system load since the sparse regions cannot fully take part in data processing. Finally, setting rs_3 to very large values (see Section 3.3.3) is not an option since the maintenance cost of the cache increases proportionally when nodes join or leave the network.

Joining the network and $cache_j$ validity: As mentioned previously, any node must maintain a consistent node cache despite the natural evolution of the network. Thus, a node joining the network must ask its neighbors to provide their node cache attested by k legitimate nodes in a region of size rs_1 centered on their location. The new node can then make the union of these caches and keep only legitimate nodes w.r.t. R_3 centered on its location. The resulting cache contains only genuine nodes and is thus valid since it has been attested by at least k nodes in a region of size rs_1 centered on the neighbors of the new node (a recurrence proof can be established).

Reusing an actor list: If there is no mechanism that prevents an attacker from reusing an actor list, then she only has to keep generating such lists until she obtains one she deems satisfactory. To counter this behavior, we put in place two mechanisms: (i) a timestamp and (ii) a limit to the number of triggered executions a node can make. With (i) we prevent any node from reusing an actor list: QLs and SLs add a timestamp to their signatures which are respectively checked by the SLs and the data sources. If the timestamp is too distant, the computation is canceled. Enforcing (ii) is possible thanks to the node cache and the k-table: the QLs solicited by Q first check if Q chose the smallest possible number of QLs (as their

node cache contains, by construction, R_1 centered on Q, they are capable of judging), thus forcing Q to choose the same QLs. They then only have to monitor and limit the number of queries Q does in a given amount of time.

Failures and disconnections: In the most complex case of node failures (i.e., unexpected disconnection) of a QL, SL or S, either RND_Q or AL cannot be computed and the protocol must be restarted (i.e., Q generates a new RND_Q). However, the probability of failures during the execution of the secure actor selection being low in our context, such restarts do not lead to severe execution limitations as mentioned above. The case of "graceful" disconnections is easier: we can safely force nodes involved in the actor selection process to remain online until its completion, thus avoiding the restarts. If a node, selected as actor wants to disconnect (or fails), the impact is mainly on the result quality since part of the results is missing.

3.3 Experimental Evaluation

This section evaluates the effectiveness, efficiency, scalability and robustness of the SEP2P actor selection protocol.

3.3.1 Experimental Setting

Reference methods. To better underline our contributions and to provide a comparison basis, we implemented three strategies in addition to the SEP2P actor selection protocol. We discarded the baseline cost-optimal and security-optimal protocols from the evaluation since the former does not provide any security while the latter is much too costly and not scalable (w.r.t. N and C) to be used in practice. Hence, we used for comparison more advanced actor selection strategies based on these protocols but using our verifiable random generation protocol with k participants (see Section 3.2.4).

The first two strategies use the verifiable random to designate the execution Setter (S) which freely chooses the actor list (as in the cost-optimal protocol). These strategies differ only in the verification process. The first one, ES.NAV (for Execution Setter, No Actor Verification) requires verifying the legitimacy of S but not of the actors. The second one, ES.AV requires, in addition, to verify that actors are genuine PDMSs. ES.AV is expected to provide better security effectiveness than ES.NAV at a higher security cost. The third strategy, M.Hash (for Multiple Hash) is derived from the security optimal protocol, but uses a DHT instead of a full mesh network. Verifiers must check that actors are genuine PDMSs and that they are "near" the random values determined by the initial verifiable random, hashed as many times as there are actors.

Simulation platform. We identified all the parameters that may impact the security and efficiency of the proposed strategies and considered all the metrics (see Table 3.3) that are worth evaluating to analyze the strengths and weaknesses of the proposed strategies, i.e., security effectiveness and cost, setup cost, scalability, robustness w.r.t. failure or disconnections. Let us note that a real implementation of the SEP2P distributed system is not very useful if we consider the above listed objectives of the evaluation. Also, measuring the scalability for very large systems (e.g., 10M nodes) with many parameters is practically impossible. Therefore, as in most of the works on distributed systems [75, 89], we base our evaluation on a simulator and objective metrics. That is, the latency is measured as the number of asymmetric crypto-operations and exchanged messages between peers instead of absolute time values.

Strategy	**Description**
ES.NAV	Execution Setter with No Actor Verification
ES.AV	Execution Setter with Actor Verification
M.Hash	Multiple Hash (with Actor Verification)
SEP2P	Proposed protocol (see Section 3.2.5)

Param.	**Description**	Values (**default**)
N	Number of nodes	10K;**100K**; 10M
$C\%$	% of colluding nodes	0.001%; 0.01%; 0.1%; **1%**; (10%)
A	Number of actors	8; 16; **32**; 64; 128; 256
α	Security threshold	10^{-4}; $\mathbf{10^{-6}}$; 10^{-10}
$\mid cache_j \mid$	Node cache size	**48** or varying from 8 to 32K
MTBF	Mean time between failure	from **1h** to 5 days

Metrics	**Units & comments**
Security effectiveness	Ratio (1 = ideal, C/N = worst)
Security cost	Number of asymmetric crypto-operations
Latency of setup cost	Number of exchanged messages and number of asymmetric crypto-operations (per minute for the maintenance overhead)
Total work setup cost	
Maintenance overhead	
Security degree (k)	Ratio (1 = ideal, C/N = worst)

Table 3.3: Strategies, parameters and metrics

This allows for a more precise assessment of the system performance than time latency, which can greatly vary in our context because of the node heterogeneity (e.g., TEE resources or network performance).

Our simulator is built on top of a DHT network. Currently, we implemented Chord and CAN as DHT overlays and use Chord for the results presented in this paper. The simulator allows to force choosing a given Execution Setter (by artificially fixing the value of RND_Q). We used this feature to obtain the exhaustive set of cases for a given network setting, each node being the Execution Setter, and then capture the average, maximum and standard deviation values for our metrics. The parameters and metrics of the simulator are described in Table 3.3. Values in bold are the default choices and their tuning is discussed throughout this section. Note that (1) the security cost is given by verifier node; (2) the latency indicates the "duration" of the protocol executed in parallel; (3) the total work indicates the cumulative number of cryptographic operations and communications during the execution of a protocol.

Security threshold value: Generating several networks and varying the security threshold α, we experimentally observed that for $\alpha = 10^{-4}$, an attacker never controls k or more nodes. However, given the importance of this parameter for the system security, we set $\alpha = 10^{-6}$ and show in Figure 3.6 the impact of choosing $\alpha = 10^{-10}$ on a small ($10K$) and large ($10M$) network. Indeed, if an attacker could master by chance k colluding nodes in a region of size $rs_1 = rs_2$, then she could completely circumvent the security mechanism of SEP2P since, for example, she can obtain k signatures from these regrouped colluding nodes for an actor list of her willing. Note that increasing α reduces the probability accordingly but increases the security cost in a logarithmic way (as discussed below in Section 3.3.3).

3.3.2 Security Effectiveness versus Efficiency

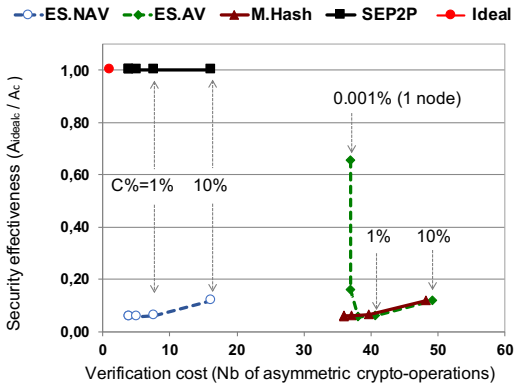

Figure 3.3: Sec. Effectiveness vs Verification

Figure 3.3 represents the security effectiveness (Y axis) versus the verification cost (X axis) for the four measured strategies and with $C\%$ varying from 0.001% to 10%. Note that the value of 10% is not realistic: it would lead to large disclosure even with an optimal random actor selection protocol, and as mentioned in Section 3.1.4, is equivalent to state-size attack. We have however run the simulation with 10% to understand its impact on the security effectiveness and cost.

Security effectiveness: SEP2P achieves an ideal security effectiveness, i.e. as good as a trusted server, independently of the number of colluding nodes. Indeed, the selection of actors is truly random, thus providing the same results as the ideal case. In addition, the security cost ($2k$) is also very low (4 to 8 asymmetric crypto-operations for $C\% \leq 1\%$). Not surprisingly ES.NAV has the same security cost than SEP2P, but the cost of ES.AV or M.Hash is much larger ($2k + A + 1$ and $2k + A$ respectively) since both must check the certificate of each actor in the list. This check allows ES.AV to have better security effectiveness than ES.NAV when C is very small ($C < A$). With respect to security effectiveness, ES.NAV, ES.AV and M.Hash are far from offering an adequate protection. Let us explain the cause for the poor security effectiveness: while RND_Q value is correctly chosen, an attacker mastering a corrupted node located "sufficiently near" from $hash(RND_Q)$ can claim to be the Execution Setter and then select a list of actors including a maximum number of colluding nodes. Here, "sufficiently near" means that it satisfies the check made by the verifiers. Note that we tuned the system parameters such that we can be "sure" to have always a node sufficiently near of any random value to allow executing the actor selection protocol for any RND_Q. The same problem arrives with M.Hash for each new random destination, thus explaining the poor security effectiveness. Hence, increasing the number of verifications or selecting each actor in a different network region does not solve the intrinsic limitation of these strategies. Note also that this behavior does not affect SEP2P. Indeed, even if the Execution Setter is a corrupted node, it cannot influence the actor list selection since it is done by $k\,SL$ (S only routes the messages between the SL).

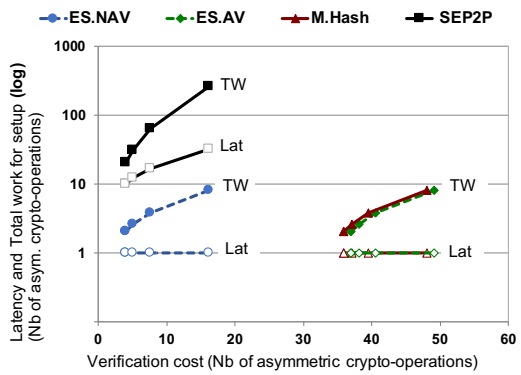

Figure 3.4: Setup asymmetric crypto-costs

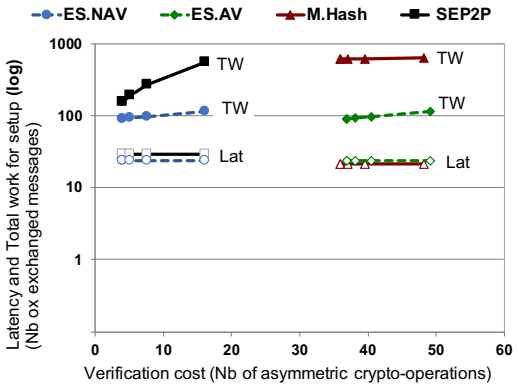

Figure 3.5: Setup communication costs

Setup costs: Figures 3.4 and 3.5 show the setup costs (Y axis in log scale) in terms of asymmetric crypto-operations and exchanged messages respectively, once more with respect to the security cost (X axis). Curves with empty symbols represent latency while plain symbols represent total work. The results show that SEP2P is the slowest in latency and has the higher total setup cost for crypto-operations. These "bad" results are the consequence of two design choices: (1) to increase the security effectiveness, we run our protocol on k *SL* nodes thus increasing the total setup cost; and (2) we voluntarily make most of the checks during the setup (e.g., checking the actor certificates or verifying their availability) in order to reduce, as much as possible, the subsequent security cost. Since the verification process is potentially performed by a (very) large set of nodes (e.g., data sources), it is in our best interest to reduce it to avoid overloading the entire system. Figures 3.4 and 3.5 illustrate this aspect: our non-optimal setup cost is balanced by an optimized security cost (and ideal disclosure in Figure 3.3). Note also that most operations are done in parallel (either by k *TL* or *SL*), thus leading to a reasonable setup latency (around 20 crypto-operations and 30 exchanged messages). We can also note in Figure 3.5 that M.Hash achieves the worst total work for setup (exchanged messages), because of the A routings in the DHT. Finally, we can remark the almost identical latency of ES.NAV, ES.AV and M.Hash on both metrics. Indeed, they all run the same initial protocol to compute RND_Q. With respect to communication, the results are also identical because all DHT routings for M.Hash are done in parallel.

3.3.3 Scalability and Robustness

We now concentrate on SEP2P to study its scalability and its robustness to node failure.

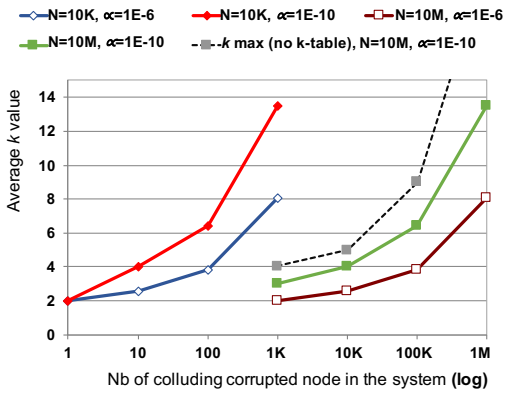

Figure 3.6: k versus C (N and α vary)

Scalability: To study the scalability, we compute the averaged k value varying C and N. Indeed, k is the main factor in the verification cost, setup latency and total work (since everything is done k times). As seen in Section 3.2.6, depending on C and N, we can compute a k-table which gives several increasing values of k with increasing region size. We have considered small (10K) to very large (10M) networks and four values for $C\%$, leading to eight different SEP2P network configurations. For each configuration, we have computed, for each node, the minimal value for k with respect to the k-table and then averaged the results. Figure 3.6 shows the average k (Y axis) versus the $C\%$ (X Axis in log scale) for several network size considering two values for α: 10^{-6} and 10^{-10}. We also plot on the same figure the value of k without k-tables (the grey curve) to highlight the benefit brought by k-tables (only shown for the large network with $\alpha = 10^{-10}$). This figure offers many insights. (1) **SEP2P is highly scalable w.r.t. N:** Indeed, k values are identical for small and large networks independently of α if we consider the percentage of colluding nodes and not the absolute value (e.g., 1% colluding nodes is equivalent to absolute values of $C = 100$ and $C = 100K$ for the small and large networks). Indeed, scaling N and C in the same proportion leads to reducing the $rs_1 = rs_2$ size accordingly. Note that with a single corrupted node, the k optimization is useless ($k = C + 1$ in that case) regardless of the α value. **(2) k increases slowly when $C\% < 1\%$:** k remains smaller than 6 even with $\alpha = 10^{-10}$. For $N = 10M$ and $C\% = 1\%$, the k-optimization reduces the number of participants in the verifiable random generation from $100K$ to 6. **(3) α has a small influences on k:** increasing α by four orders of magnitude increases k from 1 unit (e.g., $1K$ colluding nodes for $N = 10M$) to 5 units (e.g., $1K$ colluding nodes for $N = 10K$ or $1M$ colluding nodes for $N = 10M$). **(4) the k-table optimization is important:** k-tables allow reducing k by 1 unit up to 9 units (for 10% colluding nodes).

Number of actors: We also studied the impact of the variation of the number of actors (see Section 5.2.3). Overall, this results in a linear increase in the total work in terms of communications as the k SL must check for the availability of A legitimate nodes to construct their respective list of potential candidates.

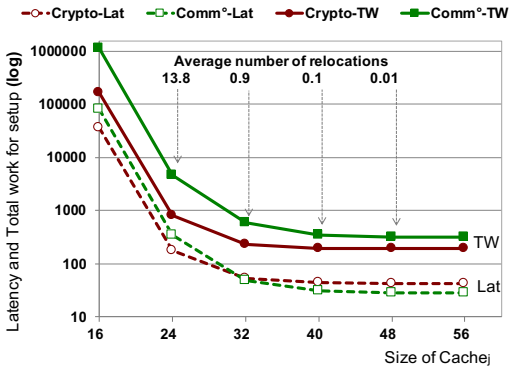

Figure 3.7: Setup costs varying R3 size

Node cache size: We now focus on adapting the node cache size to the maximum number of required actors. Our goal is to evaluate the impact of the cache size on the global performances. To do so we take a reference network with $N = 100K$, $C\% = 1\%$ and $A = 32$ and vary the average cache size on the whole network (we compute rs_3 easily dividing the cache size by N). Figure 3.7 shows the results (Y axis in log-scale). For each cache size, we simulated an execution on each node of the network and computed the average values for our metrics. Our measures show that with a very small cache, the probability of relocating the actor selection process is high (the *SL* do not find enough legitimate nodes in their cache w.r.t. R_3), which then leads to an increased latency and total work. When choosing a cache size greater than A, the query is almost never relocated (see Figure 3.7), giving better performances. This would lead to choose the largest possible cache. However, constructing such a cache also means maintaining it.

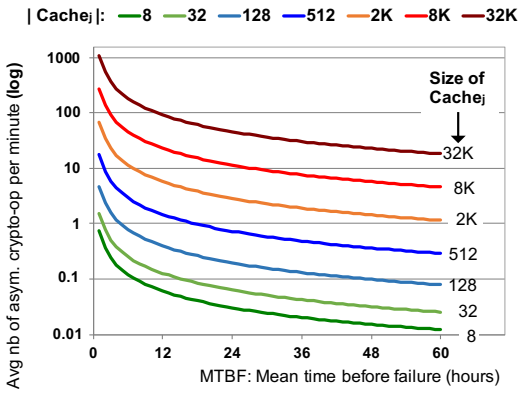

Figure 3.8: Maintenance overheads

Maintenance costs: We also evaluated the impact of the cache size in the presence of node disconnections and, more generally, the impact of disconnections. To observe it, we simulated disconnections and measured their cost depending on the size of the node cache ($cache_j$) using the default values for C, N, α and resulting k. We then considered those costs as a baseline and computed the global impact in a network where nodes disconnect (and

reconnect) every x hours ("mean time before failure" or MTBF). We represent this cost in terms of asymmetric crypto-operations (see Figure 3.8 — Y axis in log scale). The number of exchanged messages is not shown because the graphs are very similar. We also computed these metrics for large node cache sizes (up to $32K$) to confirm that full mesh networks cannot be an alternative to DHT. Our results show that an overestimated cache is excessively costly even with an MTBF of 5 days: it consumes a large portion of the overall computing power of the entire system just to maintain it up to date. With small MTBFs, the network would be probably not maintainable. Since the number of actors for a computation is likely to be relatively small (e.g., few hundred, see Section 5.2.3), we can safely set the node cache size around 512 which leads to a reasonable maintenance cost (less than 1 signature per node per minute on average for MTBF = 1 day) and with this value relocations are never triggered (see Figure 3.7).

3.4 Conclusion

Building a fully-distributed and secure query processing system based solely on PDMS nodes is arduous. First and foremost, the PDMS nodes must possess strong security guarantees: we choose to equip them with a *Trusted Execution Environment* to achieve a high level of security. At the same time, imagining that a security mechanism is perfect and building a solution on top of this assumption would be disastrous if it were to be false. Following this reasoning we introduced the *threat model* we consider: an attacker can conduct lab attacks on its PDMS nodes and fully control up to a small percentage of colluding nodes, nodes that act like *covert adversaries*. It is important to underline why we consider this percentage to be small: (i) because of the inherent difficulty of conducting a lab attack and (ii) because the discretion required to keep the nodes covert is in contradiction with a large organization (that needs visibility to attract new participants). Additionally, considering a fully-distributed protocol in a system harboring a vast quantity of colluding nodes would inevitably lead to massive leaks.

These considerations led us to define the first four requirements our solution should meet: (i) *imposed randomness* — to ensure that no attacker can influence the selection of actors; (ii) *knowledge dispersion* — to ensure that no node controls too much information; (iii) *task atomicity* — to maximize the task separation; and (iv) *hidden communications* — to protect the information from an attacker eavesdropping the network. Applying all four requirements has as a main objective to minimize the risk of a leakage and to minimize the impact of one such leak, if it were to happen.

We then introduced our first contribution: *SEP2P* a protocol that securely builds a verifiable random list of actors. By leveraging the DHT on top of which our network is built and *CSAR*, an algorithm that produces a verifiable random number as long as one honest node is involved, we showed that we can generate a verifiable list of actors and only involve a very small number of nodes — especially when we compare it to the number of colluding nodes.

Our simulation-based experimental evaluation effectively indicates that our protocol leads to minimal private information leakage, i.e., increasing linearly with the number of colluding nodes and that, at the same time, the cost of the security mechanisms depends only on the maximum number of colluding nodes and remains very low even with wide collusion attacks.

This first step lays the foundation for secure, efficient and scalable execution of distributed computations, that we will discuss in the next chapter.

Chapter 4

DISPERS

Contents

We envision three categories of queries: (i) mobile participatory sensing, (ii) subscription to information flows based on preferences or user profile, and (iii) queries over personal data contributed by a large set of individuals.

Mobile participatory sensing is used in many smart city applications for urban monitoring such as traffic monitoring (e.g., Waze or Navigon), evaluating the quality of road infrastructures, finding available parking spaces or noise mapping [94]. In these scenarios, the community members act as mobile probes and contribute to spatial aggregate statistics (density, averaged measures by location and time, spatial interpolation [94]) which in turn, benefit the whole

community. As an alternative to the classical centralized architecture, the distributed PDMS paradigm increases the privacy guarantees for the users, thus encouraging their participation. A mobile user can generate sensing data (e.g., using her smartphone or vehicular systems) which is securely transmitted and recorded into her PDMS (e.g., a home box). This way each PDMS becomes a potential data source in the system. These data can then be aggregated by a small subset of data processor nodes to produce the required spatial aggregate statistics, which can be broadcast to all the participating nodes.

Subscription to information flows based on preferences or user profile resembles RSS feeds, specific product promotions or ads. Traditionally, users subscribe to or are enrolled in a publication server that allows targeting of the interested nodes. We propose to distributively store and index profiles (or declaration of interest) in our system, in order to greatly improve users' privacy. By leveraging the DHT on which our system rely, storing and searching this information becomes straightforward: to find all the nodes matching a certain profile, a DHT search is launched, then a set of randomly selected data processors filter the results to only keep the (potentially) interested nodes. Finally, the information is sent to the selected targets.

Queries over the personal data contributed by a large set of individuals can be used, for examples, to compute recommendations or make participative studies. To achieve a high degree of pertinence and avoid flooding the system, such queries should target only a specific subset of the nodes, i.e., the nodes exposing a given user profile. Query examples are numerous, e.g., get the top-10 ranked movies by academics from Paris, or find the average number of sick leave days of pilots in their forties. The query processing is done in two steps which roughly correspond to the category (ii) combined with (i). First, the relevant subset of nodes, which match the query profile, must be discovered (category (ii)). Then, the selected subset of target nodes become data sources which supply the required data (e.g., number of sick leave days) to compute the query result (category (i)). The main differences are that only the selected nodes provide data and that the result is transmitted only to the querier node and not to the entire system.

In the rest of the chapter we thus consider queries matching the third category as it covers the other two. To arrive at our proposed solution, we first clarify the data and query model: which information is available and how we intend to query them. We then gradually construct a protocol that satisfies all the constraints we fixed. We start with a `Naive` version showcasing the feasibility of obtaining a distributed query protocol. As this `Naive` protocol has obvious limitations — an all-knowing central actor — we follow up by improving this first version and propose the `Compartmentalized` protocol that applies the *knowledge dispersion* and *task atomicity* principles. This version also being imperfect — an actor still has access to more data than needed and some nodes can be identified — we build upon it and notably apply the *hidden communications* principle to obtain the core of our final version, `Dispers`. To arrive at the final iteration, we discuss possible attack scenarios, the corresponding countermeasures we integrated and even some optimizations, eventually leading to `Dispers`. We leave the in-depth security analysis and the evaluation of `Dispers` for the next chapter.

4.1 Query and Data Model

4.1.1 Data Model

Great care must be taken when querying a potentially large number of nodes. In addition to the sensitive data manipulated, the ability to not hinder the primary purpose of a PDMS is also crucial. Indeed, preventing the PDMS to function normally would cause users to refrain from participating in the distributed computations. The random (and uniform) selection of the actors in the network already partially contributes towards that objective. A complementary measure we can take is to ensure that not all the network is involved in a computation but only a small subset. However, poorly choosing this subset can lead to uninteresting results or no results at all: imagine asking for restaurant recommendations in Toronto and none of the participants live there. It is for that purpose that a *profile* is associated to each PDMS: it is used to target relevant nodes and thus limit the amount of participants. A profile can be sensitive and as such requires protection, as explained in Section 4.3.1.

Definition 1. PROFILE. *A profile p is a set of concepts $p = \{c_1, c_2, \ldots, c_k\}$.*

Definition 2. CONCEPT. *A concept c is the concatenation of one or more metadata terms m_i describing its semantics and a value v, i.e., $c = m_1|m_2|\ldots|m_p|v$.*

Examples of concepts are: `location`|*Paris*, `sex`|*male*. Multiple metadata terms can be used to indicate a concept at different granularity, allowing for a structured organization. For instance, location at the city level: `location`|`city`|*Paris* or at the country level: `location`|`country`|*France*.

The profile is supposed to be an accurate description of the owner of the PDMS. Its generation can be achieved by different means: automatically by the PDMS according to the data it contains, manually by the owner by selecting attributes she finds fitting, or both. Beside the profile, the user may contribute with part of her stored personal data (e.g., shopping preferences, graded movies, physical activities stats, etc.) to allow for aggregated data queries from other users. We assume that the profile and the user's data are structured to allow for easy query formulation and processing.

Still, specifying the nodes that should answer a query thanks to a profile might not be enough: there is no guarantee that this filtering will be sufficient (e.g. with a very inclusive profile). To remedy this problem we **sample** the participants: we randomly select a subset and only query it. Enforcing sampling yields several benefits: (i) we achieve an overall better privacy as, in case of a leakage, a smaller subset is impacted, (ii) we remain efficient in all situations and (iii) the quality of the end result is not necessarily worse — mathematical statistics show that under certain conditions[1], an adequate sample is as representative as the entire population. Although we will not specifically come back on this step later in this chapter, since its application does not need further explanations, we want to stress here its importance: *sampling* brings additional benefits and is a key component in our approach.

Hence, to summarize, the data model we consider is comprised of two elements: the stored personal data and the profile of a user.

[1] See Hoeffding's inequality[39]for example.

4.1.2 Query Model

To understand how we propose to decompose a query, we use the following example as a basis: let us suppose that we want to know the average rating of the latest romantic movie given by people who live in Paris and who usually enjoy comedy. For conciseness, we can translate this request into a query: "average rating of the latest romantic movie as per people living in Paris and enjoying comedy".

This query exposes a *profile* to respect: "people living in Paris and enjoying comedy". If we use the *concept* terminology, this would equal:

$$\text{profile} = (\texttt{movie}|\textit{comedy}) \wedge (\texttt{location}|\texttt{city}|\textit{Paris})$$

This *profile* is the first part of a query, the *target profile*:

Definition 3. Target profile (tp). *The* Target Profile *is a logical expression of concepts indicating which nodes qualify to answer a given query.*

Defining this profile prompts us to define its corollary, a *target*:

Definition 4. Target (T). *A* Target *is a node whose profile matches the target profile.*

The following part is the data each target should provide to the distributed computation: "rating of the latest romantic movie". We call it the *local query*:

Definition 5. Local query (lq). *The* Local Query *is the expression of the query to be computed locally by each Target.*

The final part describes the function that should take as inputs the local results and provide the final result: "average" in our example. We call it the *aggregate query*:

Definition 6. Aggregate Query (aq). *The* Aggregate Query*, aq, is an aggregative expression applied over the results obtained by the Local Query.*

By "aggregative expression" we consider classical aggregate queries (e.g., average, count, min, max, top-k, group-by, etc.). More complex queries can also be considered as soon as its execution can be decomposed in a *local query* and an *aggregate query*. We leave open the query expressiveness issue for future work, the focus here being on a secure and privacy-preserving query evaluation.

Hence, in our system, we define a query as:

Definition 7. Query. *A query q is a triplet: $q = (tp, lq, aq)$.*

Let us consider three examples of query that can be formulated in our system: a closed item list query where the participants' answers concern an already defined list of items, an open item list query where there is no fixed set of answers, and a statistical query.

- **Closed item list**: "given a set of movies find their average grade as given by researchers or professors who live in London".

  $tp =$ $(\texttt{location}|\textit{London}) \wedge ((\texttt{occupation}|\textit{researcher}) \vee (\texttt{occupation}|\textit{professor}))$
  $lq =$ select grades for movies in {item list}
  $aq =$ average

- **Open item list**: "get the top-10 ranked movies by users that are researchers or professors and live in New-York".
  $tp =$ (`location`|*New-York*) $\wedge$ ((`occupation`|*researcher*) $\vee$ (`occupation`|*professor*))
  $lq =$ select top-10 in {movies}
  $aq =$ top-10

- **Statistics**: "What is the average number of sick leave day in 2017 of researchers or professors living in Paris".
  $tp =$ (`location`|*Paris*) $\wedge$ ((`occupation`|*researcher*) $\vee$ (`occupation`|*professor*))
  $lq =$ {number of sick leave days in 2017}
  $aq =$ average

Now that we have seen which data are available, the structure of a query and how both can be leveraged to constrain the number of nodes involved, we can take a look at how we can execute a distributed query in a privacy-preserving manner.

Notes on the threat model. In the previous chapter we explained that we consider an adversary capable of conducting lab-attacks on the PDMS in its possession, leading to a situation where it controls a (very) small percentage of nodes in the network.

To not confuse the reader with all the possible ways these colluding nodes can be coerced into obtaining sensitive information, we consider up until Section 4.5 a *sealed-glass* model: the colluding nodes do not deviate from the protocol, i.e. they obediently execute the tasks they were attributed, but they observe the information they manipulate. Also note that this does not exclude the possibility of spying on the communications of other nodes.

4.2 Naive Protocol

The aim of this first iteration is to assess the feasibility of simply executing a distributed query using all the data structures at our disposal. As defined in Section 3.1.1, we call *Querier* the node originating a query.

After generating its query, $q = (tp, lq, aq)$, Q has to start by finding the relevant nodes that match the *target profile* it specified. To achieve this we can leverage once again the DHT upon which the network is built: the main purpose of a DHT is to create optimized routing mechanisms to efficiently store and retrieve pairs of (`key`, `value`). Adapted to our situation and needs, the `key` is a *concept* and the `value` is a list of the IP addresses of the nodes possessing that concept in their profile. We call *Concept Indexer* every node responsible for a concept in the DHT:

Definition 8. Concept Indexer (*CI*). *A* Concept Indexer *is a node storing one or more pair·s of:* (`concept`, {`IP addresses`}).

With the addition of the *CI*, a Querier now has all that it needs to execute a query:

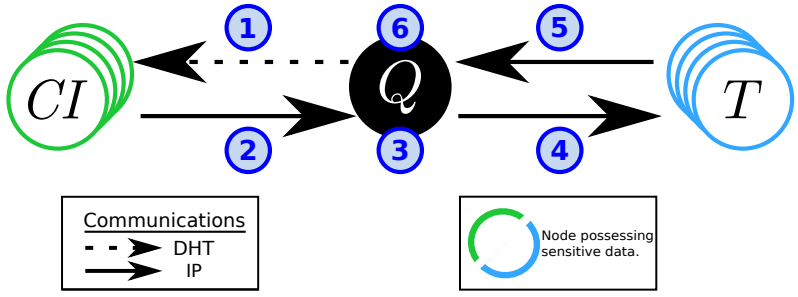

Figure 4.1: Naive Query Execution

Naive protocol

(1) The querier, Q, executes a DHT search for each concept in the target profile tp. Each concept search is routed in the DHT until it reaches the adequate CI.
(2) Each CI replies to Q with the list of IP addresses corresponding to the searched concept.
(3) Q receives the list of IP addresses for each concept from each CI. It then applies the target profile logical expression on the concept IP lists, obtains the final IP list of the targets (T) and samples it.
(4) Q contacts directly each sampled target node T and sends it the local query lq. Each target computes the local result for lq.
(5) Each target replies to Q with the local result. Q gathers the local results from the target nodes until all T have replied.
(6) Q applies the aggregate query aq on the local results to obtain the final query result and to present it to the PDMS owner.

This naive query execution protocol shows that the proposed query model can indeed be supported by our system. However, this protocol has three major shortcomings, two of them resulting from the central position of the querier. First, since any node can issue queries, a single node level attacker can use this corrupted node to issue a query and thus have access to the list of nodes matching any of the concepts of the target profile (i.e. before applying it and before the sampling), the local query results of the targets and the association between targets and local results, i.e., all the sensitive data! Second, this protocol is not efficient since the querier is a bottleneck. Third, a corrupted CI can access the list of IP addresses corresponding to the concepts it is responsible for as they are stored without any kind of protection.

These shortcomings prompt us to apply two of our design principles: *knowledge dispersion* to prevent a corrupted CI from learning the IP addresses and *task atomicity* to prevent any node from accessing a significant portion of the sensitive data.

4.3 Compartmentalized protocol

4.3.1 Knowledge dispersion

Enforcing *knowledge dispersion* to prevent the CI from directly manipulating the IP addresses can be easily achieved with Shamir's Secret Sharing Scheme[84].

Proposed in 1979, and also known as a (t, n)-threshold scheme, **Shamir's Secret Sharing Scheme** consists in dividing some data D into n pieces $D_1, \ldots, D_n$ in such a way that: (i) knowledge of any t or more D_i pieces makes D easily computable; but (ii) knowledge of any $t-1$ or fewer D_i pieces leaves D protected (not even providing any information about it).

To achieve these properties a random polynomial $q(x)$ of degree $t-1$ is generated:

$$q(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \quad \text{where:} \quad \begin{cases} a_0 = D, \\ D_1 = q(1), \\ \ldots, \\ D_n = q(n). \end{cases}$$

With enough D_i, i.e. with t or more, the coefficients of $q(x)$ can be evaluated by interpolation and D retrieved by evaluating $D = q(0)$. Whereas with less than t D_i, the coefficients cannot be computed and D remains protected.

By splitting their IP address into *shares* and assigning each share to a **different** *CI*, the PDMS nodes can greatly decrease the probability of having their IP address leaked. For instance, in a network harboring $c\%$ of colluding nodes and where the IP addresses are split into n shares (t are required to recompose the secret), the probability of obtaining the list of IP addresses associated to a concept is equal to $P = \sum_{i=t}^{n} \binom{n}{i} \times c^i$: as the concepts and the node's position in the DHT cannot be influenced and as they are uniformly distributed (see Section 3.2.2), the probability for an attacker to obtain a single list of shares for a concept is c and, as the probability of obtaining another share is independent of the probability of obtaining one, the probability of obtaining t shares among n is $\binom{n}{t} \times c^t$. The final probability is the sum of the probabilities of having $t, t+1, \ldots, n$ sets of shares. With $c = 1\%, n = 5, t = 3$ this probability is equal to $P = 0.0000100501$. Setting the values for n and t is discussed in Section 5.2.3.

Note that we do not take the instability of the DHT into account when computing this probability: although we assume that nodes in our system will, for the vast majority, stay online (the purpose of a cloud, let alone a personal one, is to be available all the time), it is likely that some of them might punctually disconnect while others might join, thus changing the shape of the DHT. We posit that this has a very limited impact on the probability and discard it.

Introducing this method to store the concepts in the DHT raises several questions: how can one know which shares go together? How are the shares securely inserted in the DHT? How are the different *CI* designated?

Shamir's Secret Sharing Scheme does not give an indication on how to know if a set of shares are related to the same secret, we thus have to provide a "marker" that signals this relationship. Furthermore, as there are possibly many shares for as many different secrets, this marker has to be *unique* per secret to not mix shares that do not belong together. We call this marker a *Concept-Target Identifier*:

Definition 9. Concept-Target IDentifier (*CTID*). *A* Concept-Target Identifier *is a unique identifier associated to all the shares that, together, recompose into an IP address. This identifier* ***must*** *be unique per concept, i.e. for any two concepts* c_a *and* c_b *possessed by the same*

```
// Index entries at CI_0.
{
    'location|city|Paris': [
        {
            'share-IP' : '1-557ea7be9d7a0fa058953ac750',
            'CTID'     : '35950928-2517-443c-a6e8-af851800b347'
        },
    ]
}

// Index entries at CI_1, indexing the same concept.
{
    'location|city|Paris': [
        {
            'share-IP' : '1-557ea7be9d7a0fa058953ac750',
            'CTID'     : '35950928-2517-443c-a6e8-af851800b347'
        },
    ]
}
```

Figure 4.2: Example of *CI* index entries

node, the CTID associated are such that $CTID_{c_a} \neq CTID_{c_b}$*; and for any two nodes* n *and* m *and any two concepts* c_i *and* c_j*, the CTID associated are such that* $CTID_{n,c_i} \neq CTID_{m,c_j}$*.*

Generating such identifier can be easily achieved with a hash (e.g. using the public key of the node, the concept and a random: $CTID_c = hash(kpub \mid c \mid RND)$). We thus assume that each node generates one such identifier for each of its concepts and associates them, correctly, to the shares — as illustrated in Figure 4.2.

Securely inserting the shares in the network is a matter of preventing an attacker from tying an insertion to a node. According to assumption 4, listening to the communications of a *CI* is possible, the only difficulty being in finding its IP address (to know "where" to listen). As we cannot keep the *CI* anonymous, we can safely posit that an attacker can, and eventually will, acquire this knowledge and is thus able to intercept its communications. This has three major implications: during an insertion the communications with the *CI* must be **anonymized**, **encrypted**, and **verified** at risk of disclosing the association (`concept` $\iff$ `IP address`). This process is illustrated by Figure 4.3.

Anonymization. By simply observing the incoming communications at the different *CI* indexing the shares of a concept, an attacker knows with certitude that the sending node possesses said concept. To hide this, nodes must employ a *proxy* to anonymize these exchanges (node P and step (1) in Figure 4.3). Still, if an attacker can intercept all the shares then it can uncover the IP address of the node, effectively nullifying this protection.

Encryption. With encrypted communications the attacker can no longer access the shares of the *CI* under its scrutiny. However this is not enough as, as shown in Section 3.3.2, an attacker can impersonate the *CI* if some of its colluding nodes are "close" enough to them

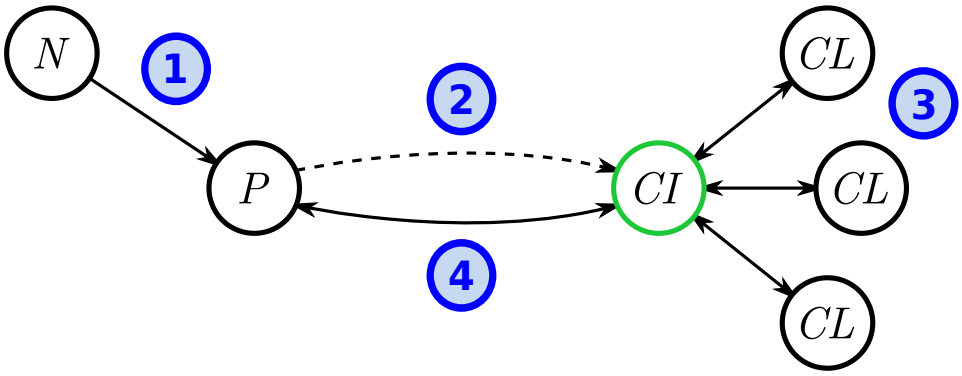

Figure 4.3: Insertion of a concept in the DHT

(one for each *CI*). An additional security measure is hence required.

Verification. To prevent an attacker from pretending to be a *CI* we propose to leverage once more the "k close neighbors" property: if k close neighbors (i.e. we have the certainty that one of them is honest — nodes *CL* in Figure 4.3) of a node are attesting the fact that it does manage the concept then this node's claim can be considered as true. Therefore, whenever a node looks up a concept it first checks the k attestations provided by the answering node before continuing (steps (2), (3), and (4) in Figure 4.3).

The only remaining problem is how to find the different *CI* responsible for a concept. In a DHT, finding out which node is responsible for a `key` is done by hashing the `key`. We propose to apply the same reasoning and to hash recursively the `key`, its hash, the hash of its hash, etc. . . to designate (and find) as many *CI* as necessary.

Hence, to summarize, by applying the *knowledge dispersion* design principle the nodes protect their association (`concept` $\iff$ `IP address`). For that purpose they: (i) split their IP address into n shares ($t \leq n$ shares being required to recompose it), (ii) store each at a different *CI*, and finally (iii) make sure to protect these communications (through anonymization, encryption and verification). Doing so greatly reduces the probability of a leak (if not completely nullifies it) as the only possible course of action for an attacker is to hope that its colluding nodes will manage the shares of the concept·s it is interested in.

4.3.2 Task Atomicity

When we look at the different steps of the Naive protocol, we can identify three roles accumulated by the querier: (i) Step (1): initiating the query by contacting the *CI*s, (ii) Step (4): finding the targets, and (iii) Step (6): executing the aggregate query.

Leaving the initiation of the query to the querier is the best option: if it had to delegate it to a random node then it would need to provide it with the necessary information to bootstrap the query and, if this random node were to be corrupted, this would disclose potentially sensitive information to an attacker. Roles (ii) and (iii), however, have to be delegated to other actors: knowing which nodes match a given profile and which results they contribute is specifically what we aim to protect. Applying the *task atomicity* design principle leads to the definition of the following new actors:

Definition 10. Target Finder. *A **Target Finder** (TF) recomposes the IP addresses based*

on the shares, applies the target profile to determines the targets, selects a sample, and sends the local query to the sampled targets.

The querier has to provide a *TF* with the *target profile* so it can apply it on the recomposed IP addresses. Yet, giving the profile as is simply moves part of the problem: a *TF* is still able to know which nodes match a given profile. A solution is to provide the *TF* with a **pseudonymized** version of the target profile: if we keep our illustrative example from Section 4.1.2, instead of receiving $(\texttt{movie}|\textit{comedy}) \wedge (\texttt{location}|\texttt{city}|\textit{Paris})$, they receive $(x \wedge y)$. Associating a concept to its pseudonym and transmitting them is done by *Q*: when it contacts the *CI* it gives them the pseudonym corresponding to the looked-up concept and the *CI* only transfer the pseudonyms (with the list of shares) to the *TF*.

Definition 11. Data Aggregator. *The role of a **Data Aggregator** (DA) is to aggregate the local results according to the aggregate query.*

Similar to the target profile, a *DA* does not have to know the nature of the data it manipulates. The querier will thus send it a pseudonymized version of the aggregate query. With our illustrative example there is nothing special to do, since computing an "average" does not disclose anything more than what the data-set itself does.

To further reduce the potential data disclosure (and to avoid performance bottlenecks), we consider having several *TF* and *DA*. The number of *TF* and *DA* are two system parameters that are discussed in the experimentation section, see Section 5.2.3. However, having several *TF* and *DA* requires adjustments.

Considering several *TF* in addition to using shares leads to a small complexity when it comes to distributing these shares to the *TF*: to determine a given target, at least t shares for each concept involved in the target profile must reach the *same TF*. Thus, the *CI* need to store, together with the share and the *CTID*, a value that ensures that they all direct the shares to the adequate *TF*. This value, called the "selector", is chosen randomly by each node and permanently associated to all its shares. The *CI* leverage it to select one of the *TF* — by computing a modulo with regard to the number of *TF*. The value of this selector should be chosen such that it is greater than the maximum number of *TF* in any query and, at the same time, small enough to favor collisions — which avoids permitting a node identification and would thus nullify the interest of using shares. We can set:

$$selector = hash(kpub_i) \mod \max(|\ TF\ |)$$

to ensure a uniform distribution.

Considering several *DA* requires the definition of another actor responsible for the final aggregation, as none of the *DA* possesses all the results:

Definition 12. Main Data Aggregator. *The role of the **Main Data Aggregator** (MDA) is to perform the final aggregation on the partially aggregated results sent by the DAs, and to transfer the final result to the querier.*

We could wonder if there are any benefits in having multiple *MDA* and organize them following a predefined hierarchy. We argue that there should only be a single *MDA* that centralizes all the partial results of the *DA* and computes the final result. This statement is motivated by two reasons: (i) as the partial results are less sensitive than the local results the number of *DA* should be maximized so as to minimize the impact of a leakage, and (ii) the *MDA* performs less operations than the *DA* (unless the number of *DA* is superior to the

number of results — a situation that should not occur), hence introducing multiple levels of *MDA* increases the latency and the total work. It is thus in our best interest to designate a single *MDA*.

To fully comply with the *task atomicity* design principle, we (obviously) impose that an actor can only play a single role in the protocol, i.e. $Q \cap \{TF\} \cap \{DA\} \cap MDA = \emptyset$.

4.3.3 Compartmentalized Query Processing

Figure 4.4 illustrates the execution of a query with the previous additions. Its process is as follows:

Compartmentalized query processing

---

(0) (Not shown in Figure 4.4) The Querier, Q, transmits the query metadata to the actors: the pseudonymized target profile and local query to the *TF* and the pseudonymized aggregate query to the *DA*.
(1) Q looks up the concept in the DHT and provides the *CI* with the pseudonym of the looked-up concept.
(2) Each *CI* checks the actors list (distinct nodes) and distributes to the several *TF*, using the selector associated to each entry in their list to ensure a correct distribution, the secret shares and their *CTID*.
(3) Each *TF* recomposes the received shares, using *CTID*, for each pseudonymized concept to obtain the IP addresses of the nodes. It applies the pseudonymized target profile to determine a subset of the targets and samples it to reduce their number. It then sends the local query to each sampled target.
(4) Each target computes the result for the received local query, randomly selects a *DA* and sends it its result.
(5) Each *DA* receives a subset of local results, applies the pseudonymized aggregate query and sends the partially aggregated result to the *MDA*.
(6) The *MDA* gathers all the partially aggregated results, computes the final query result and sends it to Q.

This compartmentalized protocol offers a welcome first round of enhancements: no actor centralizes all of the sensitive data, the querier is no longer a bottleneck, and the introduction of multiple *TF* and *DA* distributes the load, introduces parallelism and reduces the impact of leakage (more actors need to be corrupted to obtain a sensible amount of information).

Note that we did not discuss how the actors are selected: as one can imagine, leveraging a list of nodes that can be shown to have been randomly established, following SEP2P, is the best course of action and the one we evidently follow.

Nonetheless, without making a full security analysis (see Section 5.1), this protocol possesses evident security pitfalls. First, as the *TF* recompose the IP addresses before applying the target profile, if one were to be corrupted it would learn a subset of the IP addresses associated to **each of the individual concepts** mentioned in the target profile, i.e. regardless of the final selection and the sampling! Second, in accordance with the assumption 4, while the target list computation or the local result aggregation benefits from the TEE protection,

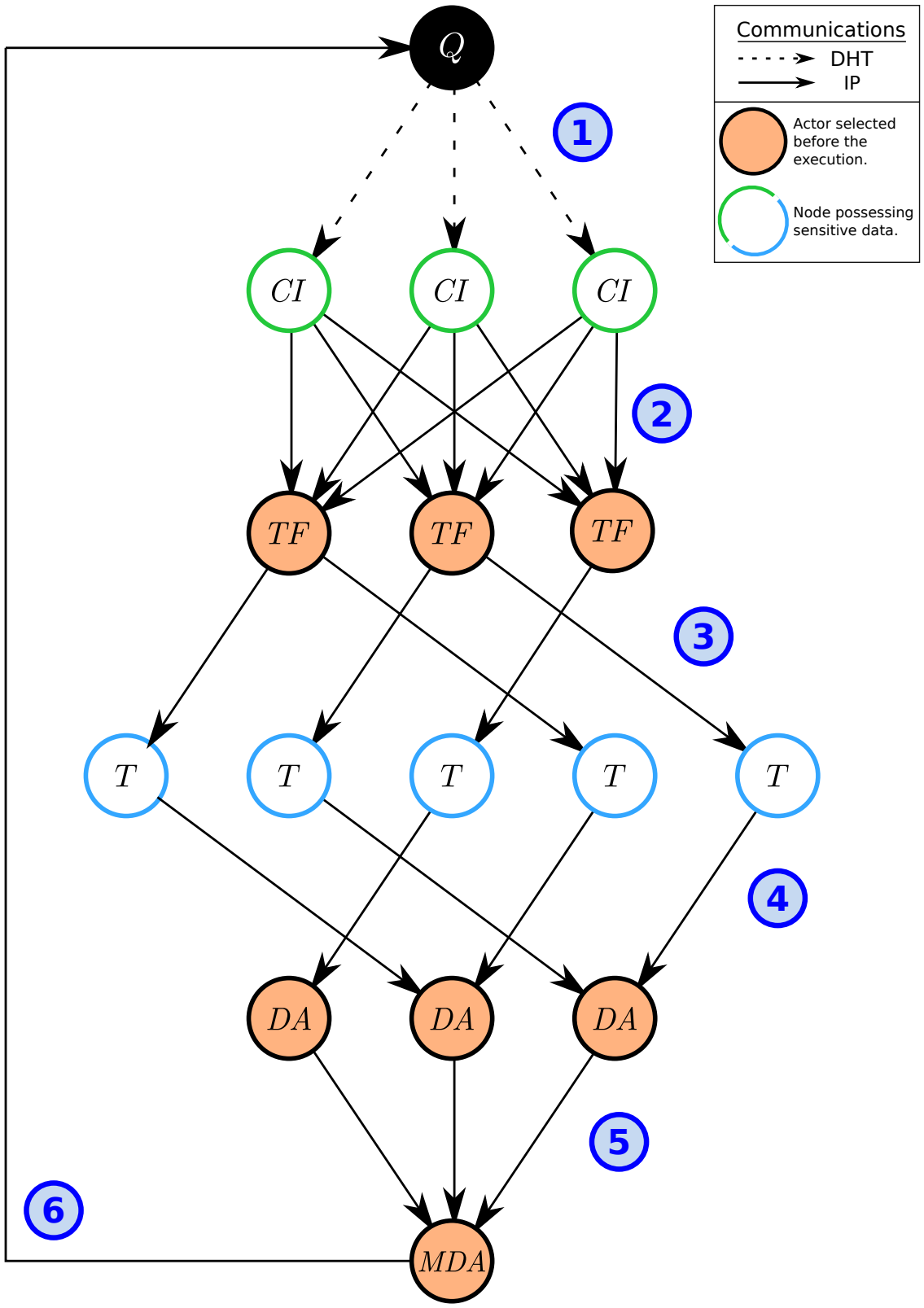

Figure 4.4: Compartmentalized Query Execution

this is not the case for the communications going to and coming from the targets: a malicious user could observe the local network traffic of her PDMS and may infer that (i) it has been selected as *TF* since it contacts simultaneously a large number of nodes and (ii) the contacted nodes are targets — all of this without conducting a lab attack! Symmetrically, a *DA* can be concomitantly contacted by a significant number of nodes, which can be interpreted by a malicious user in the same way. Additionally, none of these exchanges are said to be encrypted leaving the data completely exposed.

To solve these issues we propose in the next section the core of our final contribution: the DISPERS protocol.

4.4 DISPERS

4.4.1 Splitting the Target Finder Role

As explained, the *TF* role accumulates two problems: it has access to the IP addresses of the nodes matching any concept of the target profile, and an attacker can learn the IP addresses of the targets by simply listening to its communications. We leave aside the last problem as we answer it in Section 4.4.2 and focus here on finding a way to reduce the knowledge it has access to.

The first observation we can make is that to prevent a node from learning the IP addresses of the nodes matching not just the target profile but any of its concepts, the target profile must be applied on something other than the IP addresses. Our solution consists in having each node generate a unique pseudonym (i.e. the equivalent of an identifier), the *Target Identifier*:

Definition 13. Target IDentifier (*TID*). *A* Target Identifier *is a unique pseudonym associated to and generated by each node.*

Similar to the *CTID*, generating a *TID* can be achieved with a hash (e.g. using the public key and a random: $TID = hash(kpub \mid RND)$). However, unlike the *CTID*, the *TID* has to be associated to all concepts and, like the IP address of the node, it cannot be stored unprotected as, because it is unique, it can be leveraged to reconstruct the full profile of a node. Hence, we also protect the *TID* by applying Shamir's Secret Sharing Scheme and store it exactly like the shares of the IP addresses — i.e. the *CTID* signals the shares that belong together, a different set of shares is created per concept, and the shares are stored at the different *CI* responsible for the concept.

The introduction of the *TID* gives a glimpse of a possible separation of the *TF* role: (i) a first sub-role taking care of recomposing the *TID* and applying, on them, the pseudonymized target profile, and (ii) a second sub-role that only recomposes the IP addresses of the nodes that match. Dissociating both actions achieves our objective: a corrupted node doing (i) learns some *TID* without being able to link them with IP addresses, and a corrupted node doing (ii) learns only the IP addresses of a portion of the targets, which is the lowest possible leakage at that step and for that role. We define these roles as follows:

Definition 14. Profile Sampler (*PS*). *A* Profile Sampler *recomposes the TID based on the shares sent by the CI, applies the target profile to determine the targets, selects a sample and transfers the information of the sampled targets to a* Share Recomposer.

Definition 15. Share Recomposer (*SR*). *A* Share Recomposer *reconstructs the IP addresses of the sampled targets based on the information it received from a* Profile Sampler.

```
{
    'location|city|Paris': [
        {
            'share-IP'  : '1-557ea7be9d7a0fa058953ac750',
            'share-TID' : '1-2405989d708dfd',
            'selector'  : 7,
            'CTID'      : '35950928-2517-443c-a6e8-af851800b347'
        },
        {
            'share-IP'  : '1-4e243b1c00be99b7f24d4ee41e',
            'share-TID' : '1-d9210030db4088ad',
            'selector'  : 15,
            'CTID'      : '024b7bdd-f839-4214-8eb3-e6e9b3e2b958'
        }
    ]
}
```

Figure 4.5: New version of the CI index entries

Let us now clarify the information both roles require. To apply the pseudonymized target profile the *PS* need the *TID*. Hence, they need the shares of *TID* and the accompanying *CTID* (to know which shares go together). Applying the same reasoning with the *SR* is, however, impossible: providing them the shares of IP addresses and the *CTID* would make them able to reconstruct the IP addresses of all the nodes that match any of the concept of the target profile — which is exactly what we want to prevent.

A potential solution would be to separate the shares of IP addresses into two, give the first half to the *PS*, the second to the *SR* and make the *PS* only send to the *SR* the shares of the nodes that match the target profile. That way, without the shares of the *PS*, the *SR* cannot learn any IP addresses and we attain our objective. Unfortunately, the intrinsic nature of the shares prevents this strategy: an attacker could successively collect them (one of its colluding node is an *SR*, then, for another query, another is a *PS*, etc...) until it has enough to reconstruct the IP address. Plus, grouping compatible shares represents no difficulty if the *CTID* are transmitted together with the shares.

To not disclose the *CTID* and to prevent an attacker from blindly amassing shares, we propose the following procedure (illustrated by Figure 4.6):

Recomposing the IP addresses

(1) Each *CI* encrypts the shares of IP addresses that are to be transmitted with symmetric keys. A different symmetric key is generated per share (and encrypted with it).
(2) The *CI* send to the *PS*: the share of *TID* and the symmetric key used in the previous step. In parallel, the *CI* send to the *SR* the encrypted shares.

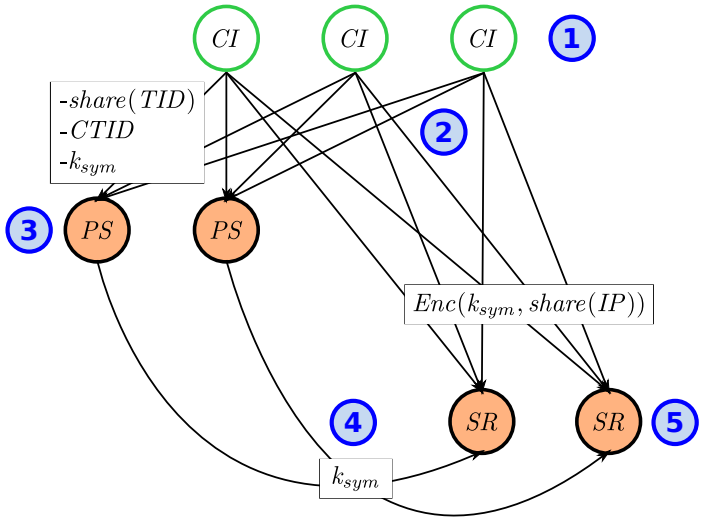

Figure 4.6: PS and SR roles

(3) The *PS* recompose the *TID*, apply the pseudonymized target profile on these and select a sample.
(4) The *PS* send the symmetric keys of the sampled targets to the *SR*. The symmetric keys corresponding to the same set of shares are bundled together.
(5) The *SR* decipher the shares and reconstruct the IP addresses of the sampled targets.

Encrypting the shares of IP addresses in step (1) prevents an attacker from collecting them. Using symmetric encryption additionally makes this operation inexpensive. Sending, in step (2), the keys to the *PS* and the encrypted shares to the *SR* separates the information: in step (5) the *SR* are only able to recompose the IP addresses of the sampled targets, because they only receive these keys in step (4).

Also, as shown in Figure 4.6, there is a one-one relationship between the *PS* and the *SR*: we arrange the list of actors to have the same quantity of *PS* and *SR* such that the *PS* do not need the selector value to determine to which *SR* they should transfer the temporary symmetric keys. They simply transfer them to the *SR* that share the same position (each in their respective list).

Hence, with this addition we solve the problem of the *TF* role, as unless both a *PS* and the corresponding *SR* are colluding, only a subset of the targets' IP addresses are leaked: the *PS* work on *TID* and do not manipulate any IP address, while the *SR* can only decipher the shares for which they received the corresponding decryption keys.

This leaves us with the communications: which exchanges should be encrypted?

4.4.2 Hidden Communications

When considering the messages exchanged during the execution of the protocol, two elements need to be considered: (i) the data itself and (ii) its metadata counterpart. Indeed, in accordance with assumption 4, an attacker can spy upon the communications of a limited set of nodes. If we assume that the querier is corrupted and has enough time to prepare, we can posit that it will listen to all the communications of the different actors: PS, SR, DA, and

MDA. We leave apart the communications from the nodes to the *CI* (to store concepts) as we have already established that they have to be encrypted in Section 4.3.1, although the conclusions drawn here are very similar.

Having access to all the communications of the actors means that the attacker knows all the sampled targets and which target produced which local result. Indeed, spying on the *SR* gives away the IP addresses of the sampled targets — every communications containing the local query is destined for a target; and spying on the *DA* gives away the association target $\Longleftrightarrow$ local result — every communication coming from a previously identified target.

Two complementary approaches are to be employed to completely seal off this information leakage: **encryption** and **anonymization**.

Encrypting the Communications

To encrypt the communications the different actors first need to be able to produce *encryption keys*. This can either be done through a shared symmetric key known in advance, or by exchanging one using public key encryption — also known as *hybrid encryption*. Given our efficiency requirement, a symmetric encryption scheme is the preferred option. However, exchanging a key while making sure that no attacker intercepts it and without relying on asymmetric encryption is difficult.

At the same time, performing asymmetric encryption is simple: the only requirement is to possess the public key of the recipient, key that can be broadcast unencrypted. For that reason, whenever it is impossible to use symmetric encryption to transmit information, hybrid encryption is performed: the sender encrypts the data using a symmetric key, the symmetric key is in turn encrypted using the public key of the recipient and the whole is sent to the recipient. Naturally, nothing prevents nodes from exchanging symmetric keys at the beginning of the protocol, as opposed to during its execution, to speed up the process.

Note that, no matter how the actors are selected, obtaining their certificates (and thus their public key) is within reach: the querier simply has to add this information to the list of actors — as that list is already transferred to all the participating nodes (see Section 4.5.7).

Hybrid encryption compared to TLS? The *Transport Layer Security (TLS)*[77] is a protocol that establishes a secure communication channel between a client and a server. This protocol is two-phased: in the first phase, the client and the server collaboratively create a secure connection and, in the second, the actual communication takes place. The first phase is the most complex out of the two as, among other, the client and the server operate a *handshake* during which they choose an encryption algorithm and a key size.

In our system, the encryption algorithm and key size are global parameters, which only leaves the actual encryption and decryption to the exchanging nodes.

Hence, which communications should be encrypted and how? Evidently, the communications between the *CI*, the *PS* and the *SR* have to: if not, an attacker spying on the *PS* and *SR* can reconstruct all the IP addresses. Similarly, the communications between the *DA* and the *MDA* are encrypted: if the final aggregation necessitates the local results instead of partially aggregated ones (e.g. to compute a median), they need to be protected. In both situations, hybrid encryption is the only option: the actors are picked randomly during the setup phase and do not know each other, requiring them to create a secure channel then.

```
{
    'location|city|Paris': [
        {
            'CTID'         : '35950928-2517-443c-a6e8-af851800b347',
            'selector'     : 7,
            'share-IP'     : '1-557ea7be9d7a0fa058953ac750',
            'share-TID'    : '1-2405989d708dfd',
            'share-key'    : '1-2fc0c178940e2b23e3',
            'key-selector' : 9
        }
    ]
}
```

Figure 4.7: Final *CI* index entries

Transferring the query metadata (the pseudonymized target profile, aggregate and local query, the concepts and their pseudonyms) from the querier to the actors and the *CI* should also be done securely: an attacker can use the metadata to infer information about the targets. For instance, a local query asking for tumor markers, coupled with the concept ($\texttt{age}| \leq 30$) and a pseudonymized target profile ($a \wedge b$) tells an attacker that the targets are less than 30 years old and suffer from cancer. Again, hybrid encryption is required to protect these exchanges.

As showed previously, the incoming and outgoing communications from the targets disclose sensitive information. The messages coming from the *SR* reveal the local query, and the messages going to the *DA* the local results. Knowing the local query and the fact that a target replied (i.e. without even knowing the local result) means that the target possesses the specific data requested; while knowing the local result might unveil the nature of the local query or, worse, disclose information regarding the profile of the target (solely based on the nature of the local result). For these reasons, we encrypt both. Choosing the type of encryption is not as straightforward as for the actors: a substantial number of nodes might be targeted, which implies as many cryptographic operations if hybrid encryption is elected. On the one hand, as the targets should not reveal themselves to the *DA* (see Section 4.4.2), there is no other option but to perform hybrid encryption. On the other hand, it is possible for the nodes to transmit a symmetric key to the *SR* without additional hybrid encryption: exactly like the shares of the IP addresses, shares of a symmetric key can be stored in the distributed index, encrypted by the *CI* and split between the *PS* and the *SR* so that only those belonging to the targets are decrypted. The only subtlety with this method is that nodes have to store a *different* key with each concept, to not recreate a second *TID*. A selector is also associated with each key so the nodes can easily retrieve the key with which some data was encrypted. Figure 4.7 gives a possible implementation although other variations, with, for instance, shares that regroup information are entirely viable.

Hence, leaving the communications open to interceptions seriously endangers the privacy of the participants, especially more so because the actors are known in advance which gives time for an attacker to prepare. Yet, limiting the protection of the communications to encrypting the data is not enough: anonymizing some of these communications proves to be a necessity.

Anonymizing the Communications: Proxies

Three sets of communications require our attention: (i) those emanating from the *CI*, (ii) those from the *SR* to the targets and (iii) those from the targets to the *DA*. Note that we focus here on which nodes are communicating, we do not consider the data being exchanged.

In (i), we have to consider the point of view of an attacker that is not the Querier and that listens to a *PS* or a *SR*. Because it is not at the origin of the query it does not know the target profile. Still, uncovering the concepts composing the target profile is achievable if the *CI* are not hidden. This problem stems from the conjunction of the use of Shamir's Secret Sharing Scheme and the DHT: because the IP addresses associated to a concept are split into several shares, as many *CI* are contacted; and because the process to designate the *CI* responsible for a concept involves the computation of a hash — that is both a deterministic operation and gives a unique value, the combination of *CI* responsible for a concept is also unique. This produces a situation where knowing which *CI* were contacted is equivalent to knowing the looked-up concept.

Cases (ii) and (iii) are identical: if an attacker (be it Querier or not) listens to the communications of either a *SR* or a *DA* it knows the targets, as they are either the destination or the source of the exchanges.

To solve all these problems we propose that the *CI*, the *SR* and the targets choose proxies:

Definition 16. Concept indexer Proxy (*CP*). *A* Concept indexer Proxy *is a node randomly chosen by a Concept Indexer which only goal is to forward data to the specified Profile Sampler and Share Recomposer.*

Definition 17. Before Proxy (*BP*). *A* Before Proxy *is a node randomly chosen by a Share Recomposer which only goal is to forward data to the specified target·s.*

Definition 18. After Proxy (*AP*). *An* After Proxy *is a node randomly chosen by a Target which only goal is to forward data to the specified Data Aggregator.*

Using proxies greatly reduces the impact of the spying of the communications of either actor: as an attacker does not know which nodes to listen to, as none of the proxies are chosen during the setup phase but dynamically during the execution, it would have to listen to the entire network to obtain the same results. Managing a wiretap on such a large scale requires resources only accessible to a state-size attacker, a threat that is outside of the scope of this work.

We do not enforce any kind of restriction on the nature of the proxies, they can be node belonging to the system (e.g. part of the node cache of any of these actors, see Section 3.2.6) or any node on the Internet — given that it is willing to do the actor's bidding. The only constraint there is concerns the *Before Proxy*: they must possess a genuine certificate so as to receive the IP address of their designated target **encrypted**. Otherwise, an attacker listening to the communications of the *SR* would learn them, which defeats the purpose of having a proxy in the first place.

Proxies compared to Onion routing? Onion routing[76, 96] is a way of routing messages over the internet that is resistant to both eavesdropping and traffic analysis. The core mechanism of this technology is to route every communication through a sequence of machines called *onion routers* where each onion router can only identify the previous

and next hop along a route.

A proxy can be assimilated to onion routing where there is only a single layer. It is thus both simpler and less expensive as there is only round of encryption/decryption to perform.

Given our current threat model, if the proxies are correctly chosen then a single level is enough. Employing "real" onion routing can be considered under other more invasive threats (e.g. an attacker able to spy on a large portion of the network).

Counting on the proxies to hide the identity of the *CI* and the targets, we now possess a framework that is able to protect the sensitive information against a "passive" attacker. Let us the detail the current state of our solution and of the query execution process.

4.4.3 DISPERS: Core Solution

Figure 4.8 illustrates the query execution process. The number of *CI* has been reduced to not make the figure too dense. A step by step description is as follows:

DISPERS Query execution

(0) (Not shown in Figure 4.8) The Querier, Q, transmits the query metadata to the different actors: the pseudonymized target profile to the *PS*, the local query to the *SR*, the pseudonymized aggregate query to the *DA* and the list of all actors (containing their certificates) to all of them.
(1) Q looks up the concepts in the DHT and provides the *CI* with the pseudonym of the looked-up concept and the list of actors.
(2) Each *CI*:
   (a) encrypts the shares of the targets' IP address and symmetric key with "temporary" symmetric keys;
   (b) determines, using the selector associated with each entry, to which *PS* and *SR* the shares should go;
   (c) encrypts (hybrid encryption) the information destined to the *PS*: the shares of *TID*, the *CTID*, and the corresponding temporary symmetric keys;
   (d) encrypts (hybrid encryption) the information destined to the *SR*: the encrypted shares of the targets' IP address and symmetric key;
   (e) asks as many proxies, *CP*, as it seems fit to transmit the information to the *PS* and the *SR*.
(3) The *CP* forward the information to their designated *PS* and/or *SR*.
(4) The *PS* decipher the shares and the *CTID*, reconstruct the *TID*, apply the pseudonymized target profile, select a sample, and send, encrypted (hybrid encryption), to the *PS* the temporary symmetric key associated to the sampled targets.
(5) The *SR* decipher the shares, reconstruct the IP addresses and their matching symmetric keys, encrypt the local query and the list of actors (symmetric encryption based on the reconstructed keys) and ask *BP* to forward the information to the targets.
(6) The *BP* decipher the IP addresses of the targets and then forward them the information.
(7) The targets decipher the local query and list of actors, apply the local query, choose a *DA*, encrypts (hybrid encryption) their result for the selected *DA*, and asks *AP* to forward the information.

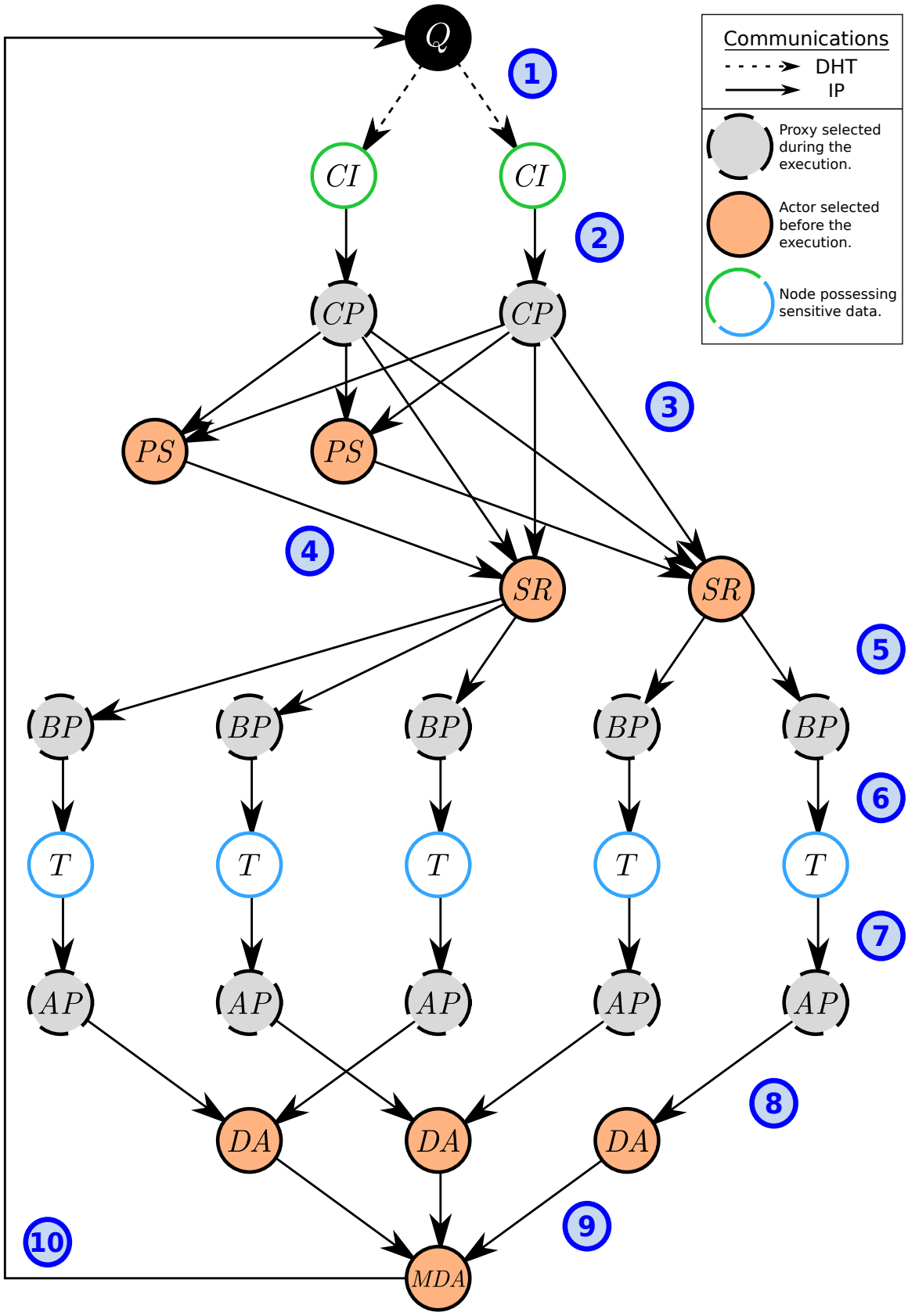

Figure 4.8: DISPERS core protocol

(8) The *AP* forward the results to their designated *DA*.
(9) The *DA* decipher the local results, apply the pseudonymized aggregate query (whenever possible), encrypt (hybrid encryption) the partial results for the *MDA* and forward them.
(10) The *MDA* deciphers the partially aggregated results, compute the final result, encrypts (hybrid encryption) it for Q and finally sends it to Q.

It is interesting to look at the information accessed by each actor with this new iteration of the protocol, summarized in Table 4.1 below. The *CP* and *AP* are omitted as their only knowledge is, respectively, the IP address of a pair of (PS, SR) and a *DA* — which are not sensitive.

Actor	Information
Q	- the (pseudonymized) target profile: concepts + pseudonyms; - the local query; - the (pseudonymized) aggregate query; - the list of actors;
CI	- the list of actors; - a concept and its pseudonym;
PS	- the list of actors; - the pseudonymized target profile; - the concepts' pseudonyms; - a subset of *CTID* (one per concept for each "node"); - a subset of *TID*;
SR	- the list of actors; - the local query; - a subset of the targets' IP addresses; - a subset of the targets' symmetric keys;
BP	- a target's IP address;
T	- the list of actors; - the local query;
DA	- the list of actors; - the pseudonymized aggregate query; - a subset of local results;
MDA	- the pseudonymized aggregate query; - the partially aggregated results; - the final result.

Table 4.1: Information accessed by the actors

We can see that each actor receives the minimum amount of information. Indeed, if only a single information is missing for any actor then they cannot perform their task: the *CI* are asked to transmit their lists and for that purpose only receive the concept, its pseudonym and the recipients (via the list of actors); the *PS* are supposed to apply the pseudonymized target

profile and only receive the necessary elements to perform that task; the *SR* have to transmit to the targets the local query and the list of actors, and receive exactly just that; and finally the *DA* are requested to compute the pseudonymized aggregate query on the local results, which is what they receive.

Additionally, as we already noted with the Compartmentalized protocol, distributing the query processing on several distinct actors increases parallelism and thus query processing efficiency. But, more importantly, DISPERS offers a *maximum degree of task separation*, i.e., inter-task, involving distinct dedicated actors for both target computation and data aggregation tasks; and intra-task, since several actors are designated for each query task.

Moreover, the task compartmentalization is complemented by a query metadata compartmentalization since only the required information is given to the query actors, and, when possible, even pseudonymized. Finally, using proxies and encrypting the communications drastically reduces the disclosure as an attacker no longer gain knowledge by simply spying on the communications of the different actors.

Hence, DISPERS sets sound foundations for privacy efficient distributed query processing. However, up until this point we have only considered attackers that opportunistically tries to access information: by starting a query and then listening to the communications of the actors or by analyzing the information coming on their compromised nodes. Considering other, more pernicious, attack scenarios requires dedicated counter-measures.

4.5 Additional Protections and Optimizations

In this section we no longer consider a *sealed-glass* model: an attacker can alter, withhold, replace or even discard the information manipulated by its corrupted nodes, all in the sole purpose of gaining more knowledge about the targets. The major consequence of this statement is that actors, in particular the *CI* and the targets, must perform **verifications**. The first verification every node must make is the integrity of the list of actors: as they manipulate sensitive information, it is mandatory that they communicate with the correct nodes. We thus assume from here on that this check is performed.

After analyzing other obvious (and problematic) attack scenarios involving a corrupted querier and then studying more subtle and opportunistic attacks, we discuss optimizations that help reduce the additional cost of the newly introduced verifications.

4.5.1 Setting and Validating the Query

Given that the Querier decides all the query parameters — the target profile, the local query and the aggregate query — it is the ideal role that a corrupted node must play in order to maximize the benefits of a successful attack. In other words, it will shape the query so as to target the nodes it is interested in and so as to learn the specific information it wants to learn about them.

A first protection we can put up is to ask for a validation of the query before it is run in the network. To do so, we establish a set of authorized queries (e.g. validated asynchronously by trusted third-parties) and we force the querier to ask k of its close neighbors to attest, through a signature, the validity of the query. As among the k close neighbors, we have an extremely high probability of having at least one honest node (see Section 3.2.3), we know that if the signatures match then the query is legitimate.

However, the verification of this signature can be somewhat problematic: if this process requires the raw query (i.e. without pseudonyms) then all the parameters must arrive to the actors performing the verifications, effectively disclosing information. Hence, even though the query is privacy-preserving, passing its parameters to the entire network is not. We thus have to find a way to sign and check the signatures without revealing the query parameters and while making sure that none were altered. Employing a *Merkle Hash Tree* for that effect solves all problems.

Introduced by Ralph Merkle in [61], a **Merkle Hash Tree (MHT)** is a data structure that can be used to produce a digital signature. As it name suggests, it is represented as a tree where the leaves' labels are hashes of data blocks and the remaining nodes (i.e. non-leaves) are labeled with the hash of their children's labels.

Checking if a data block was signed is a matter of computing the intermediary hashes, starting from the leaf and going up to the root, and verifying that the computed root matches the announced (signed) one.

Note that this verification process requires knowing only the concerned data block, since only the missing intermediary hashes need to be computed.

In DISPERS, the data blocks will be the query parameters: the concepts and their pseudonyms, the pseudonymized target profile, the local query, and the pseudonymized aggregate query.

Hence, our first additional security mechanism is to mandate the Querier to compute a Merkle Hash Tree representation of its query, ask k of its close neighbors to sign the root hash of that MHT and finally propagate the signatures and the corresponding *partial* MHT with the rest of the query parameters, to the appropriate nodes. By partial MHT we refer to the original MHT that is missing the branches corresponding to the data blocks that one must check. For example, the *PS* will receive a partial MHT where the hash of the local query is missing. The propagation of the MHT is detailed in Annex A.

Signing the query with a MHT yields another benefit that needs to be stressed out: it prevents an attacker from mixing queries. As the actors only have a partial view of the query parameters (the *PS* only knows the pseudonymized target profile, the *DA* the pseudonymized aggregate query), without this structure an attacker could have run several (validated!) queries in parallel and provided each actor with the variant it saw fit, eventually making the system execute a totally different and potentially harmful query.

4.5.2 Query Replay

Preventing a corrupted Querier from running a malicious query or a different one from what it declared, is a first step in the correct direction. Nonetheless, a corrupted Querier still has some room to maneuver. For instance, in the current state of the protocol, it can replay the same query over and over: the only check performed is the verification of the signatures of the query, so as long as these signatures are valid, the query can run. Moreover, an attacker has more incentive to replay a query if the associated list of actors is favorable, i.e. if corrupted nodes were selected as actors.

Three complementary protections are required to mitigate this: (i) associating an identifier to each query, (ii) linking the query to the list of actors, and (iii) establishing a *query budget*.

Query identifier. Providing each query with a unique identifier lets the participating nodes check if they have already received a query and ignore it if necessary. We compute this identifier as follows:

$$QID = hash(VAL \mid MHT \mid TS)$$

where | is the concatenation operation, *VAL* is the verifiable list of actors (obtained thanks to SEP2P, see Section 3.2.5), *MHT* is the root hash of the MHT and *TS* is a timestamp associated to the query.

Whenever a node is asked to perform a query related task it will first check if it has not already received that query (by comparing the *QID*) and, if not, record the *QID* before completing the task. The timestamp in this particular case is used as an expiration date: once the timestamp has expired the *QID* can be safely removed from the node's records, as an outdated timestamp renders the query invalid. This specific use is detailed in Section 4.5.3.

A beneficial side-effect of associating identifiers to queries is to enable the execution of multiple queries in parallel: by providing this identifier with all the transferred information, actors are able to tell which processing should be performed on which data.

Linking the list of actors. Associating the list of actors to the query and preventing a query replay severely limits the benefits of a "favorable" list of actors: it can only be used once and with the declared query. Lacking this limitation would be detrimental for the systems security: an attacker would simply have to keep generating list of actors until it obtains a suitable one and then reuse it indefinitely.

Binding the list of actors to the query can only be achieved when the list is created. Given how the *QID* is computed, it cannot be used for that purpose. In order to obtain a list, the Querier supplies the root hash of the MHT and its signature by k close neighbors to the list builders. Thus, when they produce the list of actors, they first check the validity of the signatures and then incorporate the elements needed to generate the *QID* to their signatures of the list of actors, effectively linking them.

Note that, with SEP2P, obtaining a "favorable" list of actors is an extremely rare occurrence. Furthermore, in our context, "favorable" means having a percentage of corrupted actors (slightly) higher than the percentage of colluding nodes in the network — the higher the difference, the less likely it is. Hence, although this situation can happen, in practice, its impact is limited.

Query budget. The main motivation behind this measure is to limit the capability of a node to launch queries. Indeed, if we retain the same scenario as for the previous point, where an attacker keeps on generating and executing queries, the leakage (ever so slightly that it is) will keep piling up, eventually leading to a tangible amount of information.

We noticeably diminish the appeal of this attack with either a fixed number of queries or a fixed number of targets (enforced by the *PS* during the sampling phase) allowed per period of time: arriving at the same amount of leaked information will require a lot more time.

Once again, enforcing the query budget can only be done when the Querier asks for a list of actors. Following the SEP2P protocol, the first step it performs is to ask its close neighbors for a random number. Instead of directly replying to its request, they ask for an index that indicates the number of queries it has done. They store that index (with a timestamp) and every time they receive a request for a random number, they check this new value against the one they have in their records. By forcing this index to only increase, the close neighbors can detect if the Querier is trying to swindle the system and effectively implement the query budget.

As a corrupted node might have more close neighbors than needed (and even other corrupted nodes among them) to do the first step of the SEP2P protocol, it has some leeway to bypass the query budget. Still, this leeway is limited: by making the neighbors exchange the values of the budget of the nodes they monitor they can detect anomalies and eventually ban the fraudulent nodes.

With these three protections a corrupted Querier cannot take advantage of the system by spamming queries or continuously generating list of actors. This concludes the most obvious threat our system faces with a corrupted Querier. However, while describing the protections we mentioned a timestamp used as an expiration date. As we will see, this timestamp is in fact a means to alleviate more subtle attacks related to timing.

4.5.3 Timing Attacks

As discussed in Section 4.4.2, with enough time to prepare an attacker can properly set up its means of attack. For instance, it can adapt its spying capabilities and focus on the actors of a query. Unfortunately, this types of attacks are not limited to spying. By postponing the transmission of key pieces of information it is possible to isolate a target, some results or both. If an *SR* sends the local query to a target sufficiently late or if a corrupted *BP* does the same directly with the local result, this target's result will be processed after the main batch, probably separately and ending up being disclosed.

For this reasons, we force the Querier to emit a timestamp when it contacts its close neighbors to start the generation of a list of actors, and we make all the nodes involved in the execution of the query check its validity. Hence, if the timestamp has expired, i.e. if the value obtained after subtracting the timestamp from the current time is superior to a threshold, then the actors will drop the query. Setting the value of this threshold is left to implementation as our simulations do not take into account the fluctuations in latency we can observe on the Internet.

There is only one remaining issue that we have to deal with, that relates to the isolation of nodes. What should happen if not enough targets contribute?

4.5.4 Targets Lower Bound

First note that this situation might happen regardless of the involvement of a corrupted actor. If the query is specific enough or if the targets have disconnected their PDMS, we might end up in the same situation.

Because of the distributed nature of our query processing, actors can only take localized decisions. Contacting the other nodes playing the same role is fruitless as, if they are corrupted, they will blatantly lie and misdirect their interlocutor into pursuing the execution. Hence, we define a system parameter expressing the lower bound each actor should check for, when processing targets' related data. This lower bound is checked by each actor: the *PS* verify the number of *TID*, the *SR* double-check with the number of IP addresses (in case a *PS* is corrupted), the *DA* check the number of local results they receive and the *MDA* after them.

The value of this lower bound should be carefully set as if it is set too high then a few actors might drop their task even though the total number of results is sufficient to guarantee privacy. Conducting this study is orthogonal to this work as it is dependent on the nature of the computation to be performed.

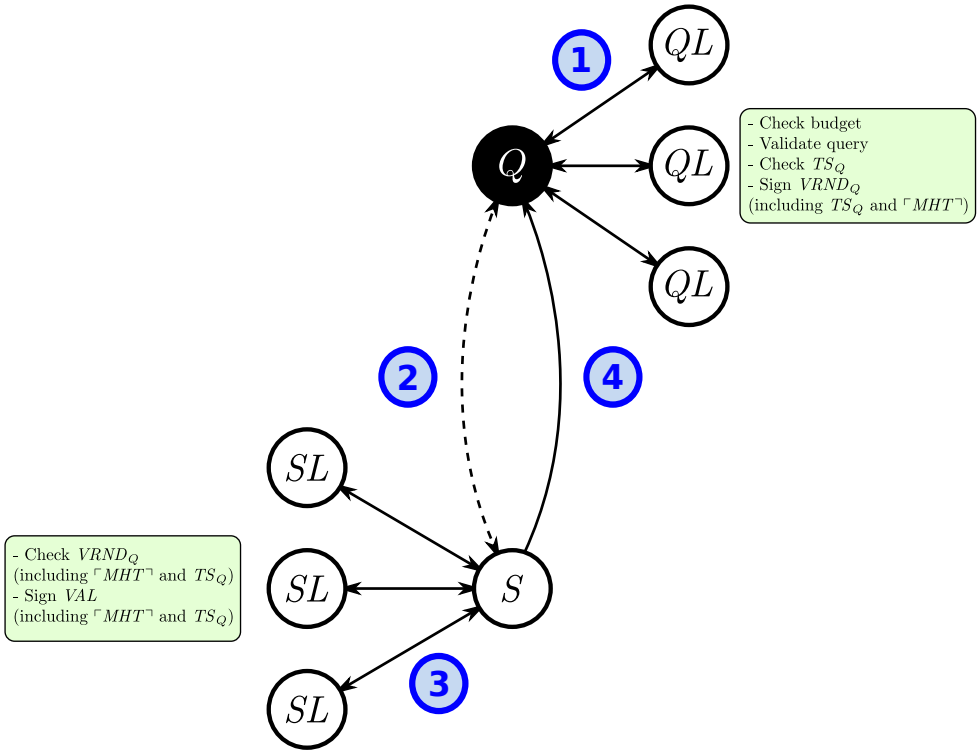

Figure 4.9: Augmented SEP2P setup phase

4.5.5 Additional Protections: Summary

Figure 4.9 and the following protocol summarize the different additions we have made to the generation of the list of actors as the different protections are enforced then. We focus here on the extra information exchanged.

Augmented SEP2P setup phase

---

(1) The Querier asks k of its close neighbors for a random number, in order to designate the Execution Setter. It supplies with its request the MHT, the query metadata, a timestamp and its current query index. If all are present and the query is valid, the *QL* proceed with the process and ultimately add the root hash of the MHT and the timestamp to their signatures.
(2) The Querier locates the Execution Setter in the DHT and forwards it, in addition to what is normally sent, the timestamp and the root hash of the MHT.
(3) The *SL* additionally check the extra elements before generating the list of actors. They add to their signatures the timestamp and the root hash of the MHT.
(4) The Execution Setter finally sends to the Querier the verifiable list of actors, containing the augmented signatures.

As we can see, when implemented, the protections translate into verifications and signatures: the *QL* enforce the budget and validate the query and the timestamp, they then add these elements to their signatures so that the *SL* can finally check the signatures before

adding the elements to their own signature of the list of actors. The different nodes involved in the execution of the query then only have to check the verifiable list of actors that regroups everything: i.e. the MHT and the timestamp.

What is important to notice is that these new verifications are performed during the generation of the list, as opposed to during the execution of the query. Once again, by doing them during the setup we reduce the workload of both the actors and the targets, which are perceptibly more numerous than the *QL* and *SL*.

Another remark we can draw is that we are able to move these verifications during the setup thanks to the "k close neighbors" property. As we will see next, this property can actually also be leveraged to optimize the execution by removing redundant verifications.

4.5.6 Optimizing the Execution

Indeed, two optimizations can be done at the *PS* and *DA* levels. It is important to first recall that (i) the Querier sends to the actors a partial MHT corresponding to the query parameter they must check and that (ii) all the actors check the list of actors to know with which node they should communicate next.

We argue that by asking their predecessors in the execution process, respectively the *CI* and the targets, to transmit the root hash of the MHT and the certificate of the next node with which they should communicate, we can skip these verifications without any supplementary cost and still attain the same security.

The rationale is that if more than k nodes transfer the same information, then this information can be trusted as at least one of these nodes is honest, despite the fact that they are not close neighbors. For this statement to hold only one condition must be met: *the nodes transferring the information must not be picked by an attacker*. If they are not picked by an attacker then it is as if the nodes where randomly picked and the same probabilities than for the close neighbors can be applied, as (on average) the proportion of corrupted nodes among them will be the same as in the entire network.

In fact, we even propose to go further than this and make the *PS* and *DA* drop their task if a single information differs from the others.

This last enhancement is the final touch we appose on the DISPERS protocol. The next section summarizes the additions before concluding this chapter.

4.5.7 DISPERS: Complete Protocol

Figure 4.10 gives a complete overview of the protocol with the new protections, associated verifications and the data exchanged. The notations used in the figure are detailed in the table below. A summary of all the information accessed by each actor is provided in Annex A as a complement.

Notation	Explanation
$VRND_Q$	Verifiable random generated by Q's close neighbors.
$\ulcorner MHT \urcorner$	The root hash of the MHT attesting the query.
$MHT_{/X}$	The partial MHT destined to X.
TS_Q	The timestamp provided by Q to its close neighbors.

VAL	The verifiable list of actors generated by the close neighbors of S. $VAL = AL + \ulcorner MHT \urcorner + TS_Q + \{sign_{SL_i}(VAL), cert_{SL_i}\}_{1 \leq i \leq k}$
QID	The unique identifier of the query. $QID = hash(\ulcorner MHT \urcorner \mid AL \mid TS_Q)$
c	A concept.
c°	The pseudonym of the concept c.
$salt_X$	The salt used to hide the hash of X in the MHT.
$CTID$	The Concept-Target IDentifier.
TID	The Target IDentifier.
k_{tmp}	The temporary symmetric key generated by the CI to encrypt each share in their list (one per entry).
T_{share}	The different shares of the targets information: $T_{share} = (IP_T + k_T + selector(k_T))_{share}$
k_T	The symmetric key generated by T.
$selector(k_T)$	The selector T associated to the symmetric key.
tp°	The pseudonymized target profile.
lq	The local query.
aq°	The pseudonymized aggregate query.
res_T	The local result produced by T.
res_{DA}	The partial result produced by DA.
$[\![\ldots]\!]^X$	Hybrid encryption at the destination of X.
$[\ldots]^k$	Symmetric encryption using the key k.

Table 4.2: Notations of Figure 4.10

Note the introduction of the *salt* in the data exchanged and in the generation of the MHT. We introduced this element to correctly hide the information: as the vocabulary of the different query metadata (concepts, local query, pseudonymized target profile, pseudonymized aggregate query) is limited, an attacker could have performed a dictionary attack against these hashes and uncover the underlying metadata. Hashing with a salt prevents this, as without knowledge of the salt building the dictionaries is extremely difficult.

4.6 Conclusion

With SEP2P we gave the foundation for a secure and efficient fully-distributed query processing system. But this foundation is indeed only a foundation and we needed to complement it with a coherent query processing protocol.

In that respect, we started by defining the data and query models to assess what and how we can query. In our system, the nodes make accessible part of their personal data and a profile, and a query is composed of three parts: the *target profile*, the *local query* and the

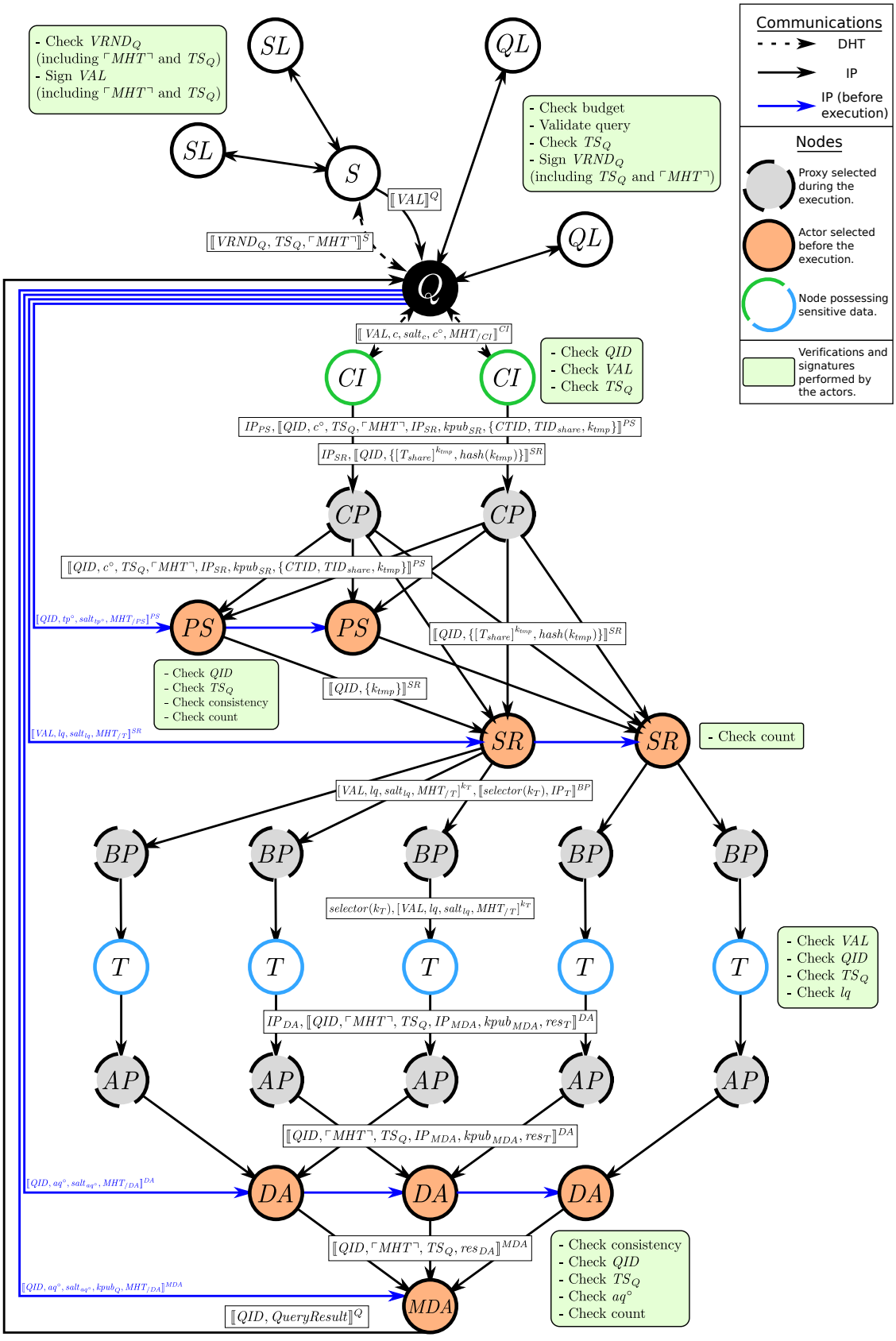

Figure 4.10: DISPERS complete protocol

aggregate query. The main goal of the (target) profile is to limit the number of participants while maintaining relevance. As this filtering may not be sufficient, we augment it with a sampling to not overflow the actor nodes.

Building on top of these element, we then showed the feasibility of a fully-distributed query protocol with a `Naive` version. Although it validated our approach, its shortcomings are far too serious for us to consider it as a viable solution. More specifically, it does not respect any of the requirements we defined in SEP2P. Hence, by successively applying each and every requirements we construct a solution respecting them: *knowledge dispersion* is enforced at the *Concept Indexer (CI)* level and requires the nodes to split the concepts they store in the DHT using Shamir's Secret Sharing Scheme; applying *task atomicity* a first time leads to the `Compartmentalized` protocol which splits the execution of a query among several *Target Finders (TF)* and *Data Aggregators (DA)*; applying *task atomicity* a second time, to split the problematic Target Finder role, leads to the definition of the *Profile Sampler (PS)* and *Share Recomposer (SR)* roles as well as the core of our solution: the `DISPERS` protocol; finally, applying *hidden communications* to address the lack of security with regards to the communications calls for the encryption of all the exchanges and the anonymization of the outgoing communications of the *CI* and the outgoing and incoming communications of the targets through proxies — *Concept indexer Proxy (CP)*, *Before Proxy (BP)* and *After Proxy (AP)*. We ultimately discussed an optimization strategy to help reduce the number of cryptographic operations performed by the *PS* and *DA* before detailing the complete `DISPERS` protocol.

Hence, iteratively applying our requirements logically leads to `DISPERS`: a fully-distributed query processing protocol based on profiles that aims at minimizing the impact of a leakage. We actually even argue that the leakage induced by `DISPERS` is under-linear with regards to the number of colluding nodes, as we show in the next chapter.

Chapter 5

Security Analysis, Evaluation and Proof-of-Concept

Contents

The first objective of this chapter is to illustrate that, under the conditions we have established, this protocol minimizes both the risk of a data leakage and the impact of one such leakage (if it were ever to happen). Following that objective, we start by providing a thorough security analysis that explains which mechanisms and properties DISPERS takes advantage of in order for us to make that claim. We then evaluate DISPERS in terms of disclosure level, setup and total work costs (in number of cryptographic operations and communications) by comparing it to other alternative solutions. We conclude this evaluation by taking a closer look to the parameters composing DISPERS and their impact.

The second objective of this chapter is to present two environments in which DISPERS has either been presented or a degraded version has been implemented: during the demonstration session of the VLDB conference held in 2019[54] and in the context of the PerSoCloud ANR project within the Cozy Cloud service as an experimental feature.

5.1 Security Analysis

In this section, we provide a general security analysis of the guarantees provided by the SEP2P and DISPERS protocols. After defining more precisely the data that could be leaked and

quantifying this leakage, we demonstrate that SEP2P guarantees with near-certitude (i.e., very high probability) a private information leakage that is, on average (i.e., for a significant number of queries), linear with the maximum number of colluding nodes of an attacker. Subsequently, we show that employing DISPERS in conjunction with SEP2P guarantees with near-certitude the private information leakage to be, on average, under-linear with the maximum number of colluding nodes of an attacker. Finally, a finer analysis of this under-linear data leakage is provided in the following section.

5.1.1 Definitions and Analysis of the SEP2P Protocol

To make this study clearer we differentiate the data present in our system into two groups: the *data-at-rest* and the *data-in-use*.

Definition 1. DATA-AT-REST. *The data-at-rest is comprised of all the information that are stored locally by the PDMS, i.e. the data owned and produced by the node and the distributed index created in the context of the DHT (the associations* `concept` $\iff$ *{*`IP addresses`*}).*

Definition 2. DATA-IN-USE. *The data-in-use is composed of all the data that are exchanged during the execution of the query. Hence, whenever a data stored locally leaves the PDMS, it becomes a data-in-use.*

Having defined and differentiated the data, we can now define the "categories" of leakage our system faces, according to their proportion.

Definition 3. LINEAR DATA LEAKAGE. *The data leakage is said to be* linear *w.r.t. some given system computation or data-at-rest indexing, if an attacker controlling c% of colluding nodes in the system can obtain at most a part of the total private data supplied by the data source nodes which is proportional to c%.*

Example. If we consider the DISPERS protocol, the data-in-use for a given computation and a given data source node i is composed of:

$$data_i = (profile, IP) \mid (local\ query, local\ result)$$

The leakage is linear if the attacker can obtain at most $c\%$ of the $\{data_i\}$ supplied by all the source nodes participating in the computation. Similarly, regarding the data-at-rest, DISPERS requires each system node to distributively store couples of (`concept`, `shares`) for each concept in the node's profile. In this case, the leakage is linear if the attacker can obtain at most $c\%$ of the lists of nodes' IP addresses associated to the indexed concepts.

Definition 4. UNDER-LINEAR DATA LEAKAGE. *The data leakage is said to be under-linear w.r.t. some given system computation or data-at-rest indexing if an attacker controlling c% of colluding nodes can obtain at most a part of the total private data supplied by the data source nodes which is proportional to* $(c\%)^p$ *with* $p \geq 2$ *(if* $p = 1$ *the leakage becomes linear).*

Lemma 1. *SEP2P leads to a linear leakage of the data-in-use.* For a statistically significant number of system computations, SEP2P guarantees a private information which is linear with very high probability with the maximum number of corrupted nodes an attacker controls.

Proof: For any given system computation (e.g., a DISPERS-like query), SEP2P provides a verifiable random list of actor nodes to process the computation. First, SEP2P guarantees

(with very high probability) that for any given system computation, the center of DHT region where the actors are subsequently chosen is randomly selected. Second, the actor node selection inside the given region among the legitimate region nodes is also guaranteed (with very high probability) to be random. This double random selection makes that, on average, the proportion of colluding actor nodes is the same with the proportion of colluding system nodes, i.e., c/N. Therefore, on average, the number of colluding actors for a computation is $A_c = A \times c/N$. This implies an average leakage of $A_c/A = c/N$. □

As shown in the above lemma, the overall leakage for many computations is proportional to the number of colluding nodes. However, using a single actor node to process a computation can locally lead to extreme private data leakage in case the randomly selected node is corrupted. To remedy this problem, DISPERS generates a list of actor nodes and evenly distributes the computation among them. This allows a better trade-off between the frequency and the size of the leakages.

Lemma 2. *Trade-off between the impact (size) and risk (frequency) of leakages.* The number of selected actors for a system computation allows to adjust between the frequency and the impact of data leakages, i.e., a higher number of actor nodes reduces the impact of leakages but increases their frequency.

Proof: Let us first consider that the actor list contains a single node. There are two possible cases: (a) the node is honest or (b) it is corrupted. In case (a) the private information leakage is 0% while in the second case it is 100%. For a significant number of computations the average leakage is thus:

$$\text{average leak} := (0\% \cdot \frac{(N-c)}{N} + 100\% \cdot \frac{c}{N}) = \frac{c}{N}\%$$

In other words, it is linear with the number of colluding nodes. However, the leakage variability in between different system computation is maximum since it alternates between 0% and 100%. Let us now assume that for any given system computation SEP2P provides a verifiable random list of A actor nodes ($A > 1$) to process the computation and that the private information is evenly distributed among the A actors. Since the actors are randomly selected, the probability to select a corrupted node is still c/N but since there are A actors, the probability of having a corrupted node in the actor list is $A \times c/N$. Hence, the risk of leakage increases with the increasing number of actors. At the same time, considering that the private data is evenly distributed among the A actors, a corrupted nodes only receives $1/A$ of the total private information required for a computation. Therefore, the impact of a leakage is proportionally decreased with the number of actors. Thus, increasing the number of actors attenuates the leakage between consecutive computations. □

Lemma 3. *DISPERS leads to under-linear leakage for all private system data.* The knowledge dispersion and task atomicity in DISPERS lead to under-linear private data leakage for both the data-at-rest and the data-in-use.

Proof: For the data-at-rest, the probability to leak the IP addresses for a given concept is $(c/N)^s$, where s is the number of shares. The leakage is under-linear for any $s \geq 2$. Hence, the probability of a leakage exponentially decreases with the increase of s. For the data-in-use, to have access to the entire private data related to a subset of target nodes, the attacker has to have at least a corrupted node among all types of actors, i.e., PS, SR and DA. But the probability for this is $(c/N)^3$, so largely under-linear. We should also stress that even in the very rare cases in which this happens, the attacker does not have any guarantee to be able

to reconstruct completely the proportion of private data due to additional mechanisms (e.g., several actor nodes per task level, anonymized communications). □

While the private information leakage for the data-at-rest is simple to quantify given the uniform data distribution provided by the DHT and the number of Shamir's secret shares required to obtain the private data, the leakage regarding the data-in-use requires a more in depth analysis.

5.1.2 DISPERS

Let us take a closer look at the data-in-use that can be leaked during an execution of the DISPERS protocol. What is important to notice is that the private information concerns the targets in relation with the query $q = (tp, lq, aq)$, and more particularly with the target profile and the local query. These associations are summed up as follows:

1. Knowing the target profile and the fact that a node is a target (via its IP address), is the most sensitive data leak in DISPERS since the attacker learns the association profile $\Longleftrightarrow$ node. A lesser version of this leak is the ability to associate a concept to a node.
   Also note that this opens up the way for a subsequent identity attack by associating the IP address to a "real" individual.
2. Knowing the local query, the fact that a node is a target, and its local result is another sensitive data leak in DISPERS since the attacker can associate a local result to a node and qualify it.
3. Knowing the *TID* and the IP address of a node can lead to subsequent disclosures: the target profile or some concept·s. This is due to the fact that there is a bijective function between it and the IP addresses.
   Similarly, this reasoning can be applied to the *CTID* and the symmetric key (used to encrypt the communications between the *SR* and the targets). The main difference for both elements resides in the possible resulting disclosures: knowing the *CTID* can lead to the *TID* and other *CTID* while the symmetric key is only directly tied to the IP address.

We do not mention knowing a local result in combination with the aggregate query as unless both are extremely discriminative (which should be avoided thanks to the validation of the query by a third-party authority), the disclosure will be severely limited.

Hence, as we can see, the leakage during the execution of a query revolves around the ability of an attacker to put pieces of information together. To do so, corrupted actors must participate in the process. In the following, the one/two/three level leaks refer to the number of actor groups that have been hacked, i.e. a level one leak means that at least an actor in the studied group is corrupted and a level two means that at least an actor in each group is corrupted. Our analysis focuses first on the base collusion attack including the colluding hacked nodes and then goes further by considering the additional leakage that can arrive following a communication spying attack (i.e., which nodes are communicating and what information is transmitted in clear). Finally, we conclude with the analysis of pure communication spying attacks.

We also distinguish in our analysis between the **directed attacks** (i.e., in which the initiator of a query is a corrupted node) and **opportunistic attacks** (i.e., in which the initiator of a query is a honest node). In a directed attack, the attacker has direct access to the query metadata since it is the query initiator. However, with a query budget, the extent

of directed attacks can be limited. In a opportunistic attack, the attacker does not have direct access to the query metadata, but tries to infer it if any of its colluding nodes is selected during the query processing. Also, the extent of opportunistic attacks is much larger since it affects most of the system queries.

One Level Leaks

1. *PS* — **directed and opportunistic attack**: the *PS* have access to a portion of the *TID*. Using a two level attack it can, with certainty (directed) or with a certain probability (opportunistic), additionally associate the target profile.

2. *SR* — **directed attack**: the associations profile $\Longleftrightarrow$ IP address can be leaked with a probability c/N. This is the most important private data leakage because of both the data sensitivity and the significant probability to happen. The workarounds are (random) sampling and query budget. Note that if the attacker controls c nodes, her query budget is $c \times$ query budget.

3. *SR* — **opportunistic attack**: The attacker knows part of the sampled targets, the local query but not the target profile. The local query might indirectly leak information about the selected nodes' profile but the inference is limited. A collusion with other node·s is required to fully leverage the information (see Two level leaks below).

4. *DA* — **directed and opportunistic attack**: in the directed case, the *DA* knows the local query and some local results but does not have direct access to the targets' IP addresses thanks to the after proxies (*AP*). Hence, associating a result to a target is as hard as de-anonymizing the communications coming to the *DA*.

   The opportunistic attack is similar in that the same conclusion can be drawn with the single difference that the attacker does not have access to the local query.

We remark that communication spying does not bring any benefit to an attacker: the different proxies prevent the identification of the targets or the concepts composing the target profile (via the probably unique combination of *CI* indexing each concept).

Most importantly, we note that except for attack 2, possessing a single corrupted actor does not yield any, directly, fruitful information: in 1, the attacker learns some *TID* without knowing any of the concepts they match; in 3, a *SR* only learns the IP addresses of a subset of the sampled targets, also without knowing their profile; in 4, the attacker accesses some local results without knowing where they came from.

Two Levels Leaks

We focus here on the most problematic combinations, even though the other are equally probable their information leakage is less beneficial as detailed afterwards.

1. *SR* + *PS* — **directed and opportunistic**: the attacker gets a part of the list of nodes matching any concept of the target profile. *For this attack to work, the PS and SR need to be a matching pair.* If the attack is directed then the attacker additionally knows which concepts each node matches.

2. $SR + DA$ — **directed and opportunistic**: the attacker gets a probabilistic association (i.e. inaccurate) between IP addresses and local results, knows the local query and the pseudonymized aggregate query (and the target profile in case of a directed attack). The uncertainty is due to the random selection of DA by the sampled targets: they can pick an honest DA instead of the corrupted one·s. Hence, once again, precisely associating a local result to a target is as hard as de-anonymizing the communications coming to the DA.

3. $AP + DA$ — **directed and opportunistic**: the attacker knows the IP address of (a very small number, if not a single) sampled targets that chose the AP *and* the DA. Consequently, the attacker knows these targets' local results and the pseudonymized aggregate query. However, the probability of this event is extremely low considering that the AP are randomly chosen and not limited to nodes in the system.

   A directed attack adds the remaining query parameters.

Among the other possible combinations there are those involving a CI or a CP: this situation is only profitable if the querier is not corrupted as, otherwise, the knowledge provided by both is redundant with the target profile since the CI gives a concept to the attacker and a CP gives an indication about a concept.

Corrupted BP and AP bring the IP address of a target. However, as they do not know for which query they are playing that role (the only information they have access to is to which node they should forward), if no other corrupted node provides them with complementary knowledge, they cannot benefit from it. Hence BP and AP must be paired with actors. At the same time, unless these actors are in direct contact with them, they have no certainty that they participate in the same query. This implicates that an AP has to be paired with a DA and a BP with a SR. Since we have already studied the former, we focus on the latter. Fortunately, this pair is redundant: they both learn the IP address of the same node·s.

As we can see, in some rare cases, a two levels attack can lead to significantly more information leakage than a one level attack — especially when the querier is corrupted (but the occurrence is even rarer). For instance, in 1 the leaked IP addresses are obtained before applying the target profile and the sampling. However, the probability for this attack is very low ($\ll (c/N)^2$) since the attacker has to obtain a "perfect" configuration with a matching pair of PS and SR. Hence, the increase in the leakage level is largely compensated by the decrease in the probability of it happening.

Three Levels Leaks or Higher

Such attacks are not interesting for two reasons. First, compared with a two levels directed attack involving a SR and a DA, it does not offer much additional benefits — a corrupted DA might bring the exact source of a local result but, as discussed earlier, the probability and the extent of this leak are severely limited. Second, the probability to happen is extremely low: $(c/N)^3$ or lower. Given the low probability, the one level and two levels leakages become much more important due to their much higher frequency. Therefore, we choose to omit such attacks in the discussion below.

Discussion

Two levels leaks have a probability of occurring that is lesser or equal to $(c/N)^2$, which is much smaller than one level leaks. A directed attack involving a matching pair of *PS* and *SR* leaks more data than a directed attack with only a single *SR* because of the lack of sampling and filtering via the target profile. Nonetheless, given the much lower probability this scenario can be considered as secondary. Combining three or more actor types does not bring much more benefit for the attacker — only a trio of *SR*, *AP*, and *DA* can theoretically bring more — while the probability continues to drop drastically.

Hence, the first major leak is at the *SR* side. To limit it, DISPERS employs two mechanisms: (i) the *query budget* associated to each node renders the directed attacks a lot less frequent. Combined with the probability (c/N) for the attacker to have a corrupted node selected as *SR*, the actual leakage plummets. (ii) For the opportunistic attacks, that are "unlimited" for lack of a global query budget, DISPERS employs anonymized communications from the *CI* to the *SR* (through the *CP*) effectively preventing an attacker from associating a profile to the IP addresses it accesses.

Another important leak could happen at the *DA* side. The attacker could associate some local data to a node's IP address. Though it is possible, managing this attack is extremely hard because of the anonymized communications between the targets and the *DA*: uncovering an association requires spying a vast amount of nodes and hope that some *AP* are among them.

In conclusion, we observe from this detailed analysis that the global leakage is clearly underlinear with the DISPERS protocol. The leak that is closest to a linear leakage regards the association target profile $\Longleftrightarrow$ sampled target's IP addresses with colluding *SR* and directed attacks. However, as we have pointed out, this association represents only a fragment of the total private data, as it happens after the sampling and within a budget. Additionally, decomposing the task of finding the targets for a query with several actors (CI, CP, PS, SR) significantly limits the disclosure of this part of the private data. Similarly, anonymizing the incoming and outgoing communications of the targets produces a separation between the target finding task and the local data aggregation task, making it complex for an attacker to associate the local results with eventually disclosed targets' IP addresses.

5.2 DISPERS Security and Performance Evaluation

In this section, we provide an evaluation of the security and performance of the DISPERS protocol. We start by evaluating the security of DISPERS and show that it attains an optimal security in our context and under a realistic threat model. We then introduce two alternative methods as competitors of DISPERS and continue with a detailed evaluation of the performance of the three methods with a particular focus of the DISPERS protection mechanisms and their respective cryptographic and communication costs. Finally, we conclude by presenting two proof-of-concepts of DISPERS.

5.2.1 Security Evaluation

Table 5.1 summarizes the average leakage each solution creates. The first two column qualify the type of attacks: if the Querier is corrupted ("Directed?") or whether the attacker is eavesdropping on the network or not ("Spying?"). The different strategies we consider are as

follows: `Naive` corresponds to the very first query processing protocol we described (not enhanced); `CQP` stands for *Compartmentalized Query Processing* and uses all the actors described in DISPERS (PS, SR, DA, MDA) without the setup phase (i.e. the Querier chooses the actors as it pleases), any encryption or proxies; and `DISPERS` corresponds to the full version of the DISPERS protocol.

Directed?	Spying?	Strategy	Target	Results	Association
×	×	`Naive`	0%	0%	0%
✓	×	`Naive`	100%	100%	100%
		`CQP (+SEP2P)`	c%	c%	c%
✓	✓	`CQP (+SEP2P)`	100%	100%	100%
		`CQP (+SEP2P +ENC)`	100%	c%	c%
		`CQP (+SEP2P +PROXY)`	100%	100%	$\leq c^2$%
		`DISPERS`	c%	c%	$\leq c^2$%

Table 5.1: Average disclosure per strategy

Between parenthesis are indicated the additional mechanisms DISPERS leverages: `+SEP2P` tells that the list of actors was generated using SEP2P, `+ENC` signals that all the communications are encrypted, and `+PROXY` signifies that the different proxies are employed to hide the communications of the *CI* and the targets. This also means that DISPERS is equivalent to `CQP (+SEP2P +ENC +PROXY)`.

The last three columns show the leaked information: "Target" stands for the IP addresses of the targets, "Results" for their local results and "Association" means that the attacker can tie a result to a target. Note that whenever an attack is directed, i.e. when the Querier is corrupted, **the profile is automatically leaked**. Hence, whenever a target is uncovered the attacker already knows the profile she matches. Obviously this situation is the worst possible, that is why we focus on it for this study.

The first element we notice is that only the `Naive` strategy achieves the ideal disclosure when the Querier is not corrupted and the attacker does not spy on the network. This result is easy to explain: an honest node concentrates all the information and protects it. Unfortunately, this strategy is a double-edged sword as, as shown in the second line, if we consider that the Querier is corrupted then the disclosure is disastrous: everything is leaked. More importantly this scenario requires a single corrupted node and is thus easy to achieve.

We then see that if the attacker is not spying on the network the disclosure is near-minimal: the attacker accesses, on average, c% of all the sensitive data — the IP addresses, their results and the associations. Once again, this near-optimal protection holds only if the attacker does not spy on the network, as if it does then all the sensitive data are disclosed: none of the communications are protected and an attacker learns everything. Although, this scenario is not as easy to set up as corrupting a single node, it still does not require corrupting a large amount of nodes.

This leads to the final lines of the table. The main purposes of the strategies `CQP (+SEP2P +ENC)` and `CQP (+SEP2P +PROXY)` is to show their synergies when they are applied together in `DISPERS`. Encrypting the communications prevents an attacker from learning the local results and the associations (result $\Longleftrightarrow$ target) but not from learning all the targets' IP addresses. Anonymizing the communications of the targets (in this scenario anonymizing the

communications of the *CI* yields no benefits as the Querier is corrupted and the target profile has already leaked) effectively prevents an attacker from associating a result to a target but, as none of the communications are encrypted, it still learns their IP addresses and results. Note that, in this case, the leakage of associations is $\leq c^2\%$ and not $c\%$ since, to uncover an association, the attacker has to have a corrupted *DA* and *BP* and "hope" that a target randomly chooses both (which, in practice, gives an average leak that is much smaller than $c^2\%$ but impossible to generalize).

As `DISPERS` leverages both strategies we can see that it achieves the minimal leakage under a realistic threat model: $c\%$ of the targets' IP addresses, $c\%$ of the local results and $\leq c^2\%$ of the associations (result $\Longleftrightarrow$ target). Furthermore, these results must be nuanced because of two additional protections we set: the query budget and the sampling. With the query budget the directed attacks are less frequent: up to $c \times$ query budget (assuming the attacker controls c nodes). With the sampling, whenever the disclosure of the targets' IP addresses occurs at a *SR*, only the sampled subset is leaked.

5.2.2 Considered Alternative Methods

To underline the importance of the security mechanisms in DISPERS and their impact on performance, we consider two alternative protocols to evaluate queries in our system. Since to our knowledge, no alternative protocol for DISPERS exists, we consider as competitors two query protocols "derived" from DISPERS. We only focus on the query protocol (i.e., the protection of the data-in-use) since we consider that the solution we propose for the data-at-rest (i.e., employing the Shamir's Secret Sharing Scheme in conjunction with the distributed DHT storage) is much less questionable due to its simplicity and effectiveness.

Naive protocol. The first protocol (see Figure 5.1) is based on the naive protocol introduce in Section 4.2. This protocol has the advantage to propose a simple solution for query processing in our system. However, to prevent data leakages in opportunistic attacks, we enhance the initial naive protocol with protection techniques inspired from DISPERS: *CI*s have to provide k attestations to Q from their close neighbors to prove that they are indeed the legitimate holders of the searched concepts and thus avoid leaking the searched target profile; the communications between Q and the *CI* and between the targets and Q are encrypted to avoid spying; and finally, the different proxies (CP, BP, AP) are present, in order to protect the identity of the *CI* and the targets from an attacker spying on the network.

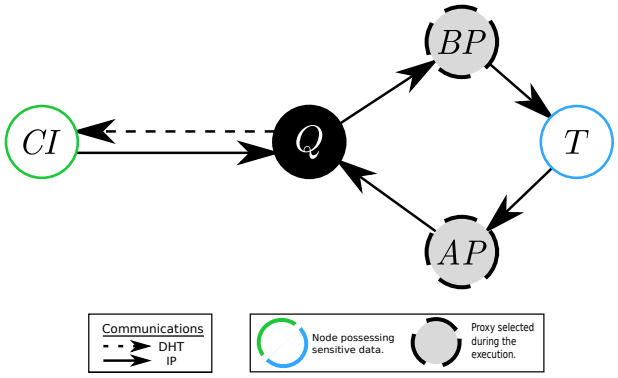

Figure 5.1: Enhanced Naive Query Execution

Parameter	Value	Parameter	Value
Security margin (α)	10^{-6}	Shamir degree	$n = 5$ $t = 3$
Number of nodes	$1,000,000$	Percentage of colluding nodes	1%
Number of concepts in the target profile	3	Number of targets (sampled)	2000
Number of PS, SR	32	Number of DA	32

Table 5.2: Simulator default values

Security-wise, we expect this protocol to be extremely risky since hacking a single node allows the attacker to access large amounts of private data. Performance-wise, we expect this protocol to have a good overall cost (i.e., total work) but large latency (since Q acts as a bottleneck).

Ideal security and cost-optimal protocol. The second protocol is based on a simplified version of the DISPERS protocol. The idea is to have an "ideal" protocol both from the performance and security viewpoints. Regarding performance, we need a distributed protocol to avoid bottlenecks as with the naive protocol. For this reason we employ a version based on DISPERS. Regarding security, the objective is to have a protocol that reduces to a minimum the security overheads. To this end, we consider that the system contains an oracle node capable of supplying a random list of actors. This way the relatively costly setup phase of DISPERS (i.e., the SEP2P protocol) is avoided. This leads to the following protocol as depicted in Figure 5.2.

Obviously, having such an oracle node clearly contradicts the fully-distributed aspect of our work. This kind of protocol is however interesting since it offers a lower bound for the security overhead in our system and thus offers a better perspective on the (necessary) security overhead in DISPERS implied by the covert adversary attack model.

5.2.3 Performance Evaluation

Notes regarding the metrics. We measured with our simulator the number of exchanged messages and the number of asymmetric cryptographic operations (as with SEP2P). Still, in order to better grasp the measurements, it is important to note the following points. (i) The number of messages does not include the messages resulting from the creation of the secure channel or the acknowledgments. (ii) Regarding latency, we consider that we can send a large number of messages in parallel. (iii) The number of asymmetric cryptographic operations takes into account the creation of a secure channel. More specifically, it counts two operations for this: one to encrypt the session key and another to decrypt it. (iv) We assume that these cryptographic operations cannot be executed in parallel and are thus included in the cryptographic latency.

The default configuration of our simulator is described in Table 5.2.

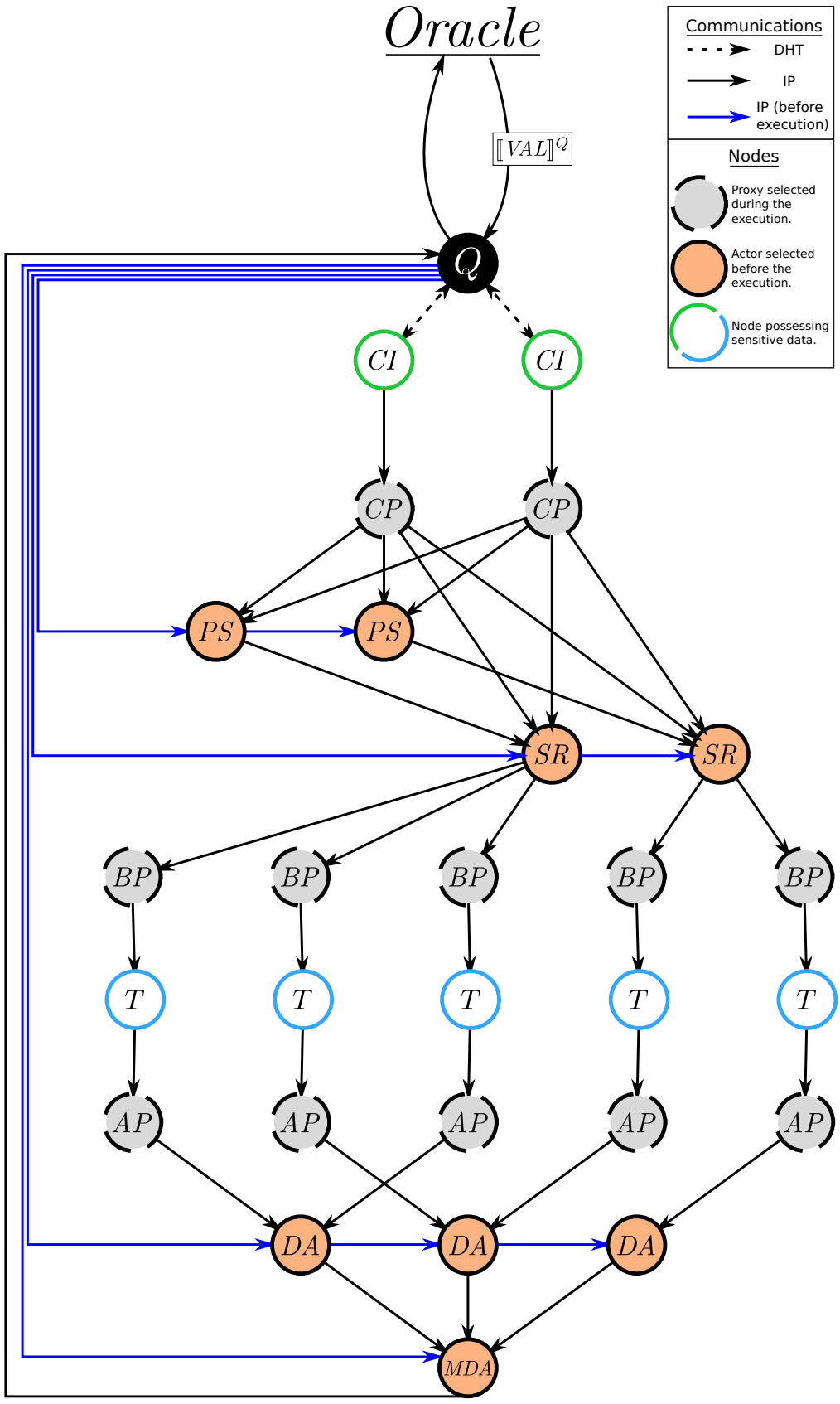

Figure 5.2: Oracle Query Execution

Influence of the Number of Colluding Nodes

Figure 5.3 below shows the cryptographic latency in the default configuration, we only varied the number of colluding nodes, C from 10 to 100,000 (the last value meaning having 10% of colluding nodes which is unrealistic, but interesting to see the trend). On the one hand, we observe that the enhanced Naive strategy performs poorly because almost all the work load is assumed by the Querier. At the same time the latency stays (almost) constant when C increases because this strategy does not perform any verification (apart from the CI but the impact is negligible). On the other hand, the Oracle strategy and DISPERS both have a relatively low latency. The difference between the two resides in the setup phase which has a minor impact. Similarly to the enhanced Naive strategy, these values do not depend on the number of colluding nodes as the verifications are done in parallel by the CI and the sampled targets.

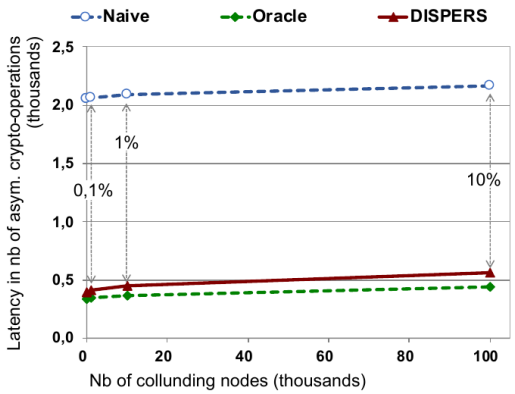

Figure 5.3: Crypto. latency vs C

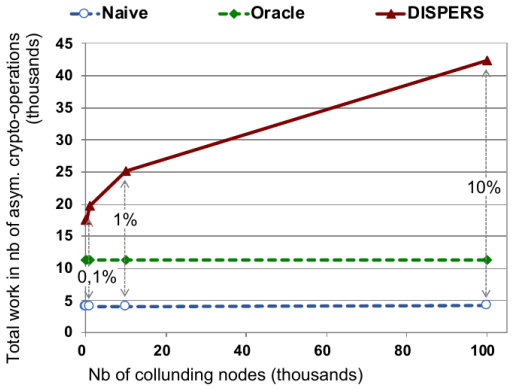

Figure 5.4: Crypto. total work vs C

Figure 5.4 presents the total work with regard to the number of colluding nodes. Unsurprisingly, DISPERS has worst performances than the Oracle strategy which is itself worst than the enhanced Naive strategy. This good result has to be put into perspective: the enhanced Naive strategy is extremely weak against a single corrupted node, since if the Querier is corrupted all the sensitive data are leaked. We observe that the difference in total work between the enhanced Naive strategy and the Oracle is of, roughly, 6,000 cryptographic operations, which is 3 times the number of sampled targets. This difference is relatively easy to explain: in the enhanced Naive strategy, the Querier has to create a secure channel with the BP to hide the address of the sampled targets. The latter, however, can use their symmetric key (k_T) to transmit their local results. Hence, only two asymmetric cryptographic operations per target are required in the enhanced Naive strategy. In the Oracle strategy, the sampled targets have to additionally establish a secure channel with the DA (via the AP) and they have to check the signature of the list of actors produced by the Oracle. This adds up to 3 supplementary asymmetric cryptographic operations per target.

DISPERS has the same behavior as the Oracle strategy but the CI and the sampled targets additionally have to check the k signatures of the list of actors, VAL, and the k certificates of the signatory nodes, totaling $2k$ operations per target instead of 1. For instance, with $C = 1,000(0.1\%)$ we measured (see Section 3.3.3 and the k-table) that the average k is equal to 2.6. It goes up to $k = 3.9$ with $C = 10,000(1\%)$. This explains the difference between the Oracle strategy and DISPERS, and highlights the importance of minimizing k.

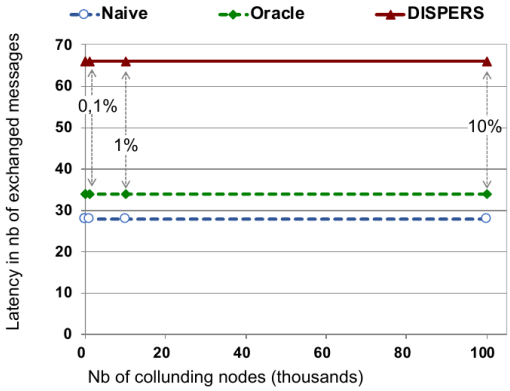

Figure 5.5: Comm. latency vs C

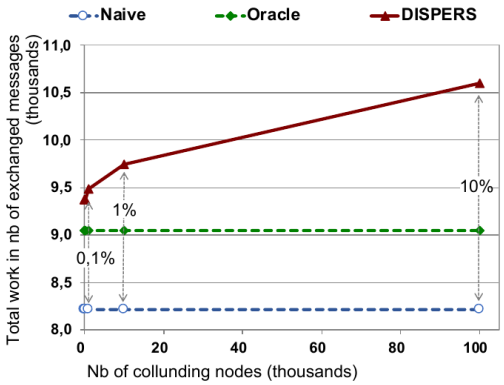

Figure 5.6: Comm. total work vs C

Figure 5.5 presents the latency in terms of number of messages. We observe that it is independent of the number of colluding nodes. Indeed, this latency is only related to the structure of the execution. The security degree, k, influences the number of QL and SL required to generate the list of actors VAL but all these communications are done in parallel. Overall, the latency in terms of number of messages is low, notably because the DHT is only solicited once (to locate the CI) in the enhanced Naive and Oracle strategies, and twice for DISPERS (the Querier has to locate the Execution Setter (S)).

Regarding the total number of messages exchanged during an execution, as shown in Figure 5.6, we notice that the differences are not as pronounced as for the total number of asymmetric cryptographic operations. This is explained by the fact that most of the exchanges occur "around" the sampled targets. For each target, and for the three strategies, the communication scheme is: $BP \rightarrow T \rightarrow AP \rightarrow DA$(or Q), i.e. 4 messages per target, totaling for $8,000$ messages solely for this part of the execution. The remaining messages correspond to the setup phase for DISPERS (and this number depends on the value of k, which explains the increase when $C = 10,000$), the communications from the DA to the MDA for DISPERS and the Oracle strategy, and when the CI are looked up (for the three strategies).

Hence, to conclude this preliminary study, let us first remind that the enhanced Naive strategy is extremely dangerous (unless a TEE is unbreakable), and that the Oracle strategy is unrealistic: only DISPERS is actually applicable in a real world scenario.

Nonetheless, we noted as expected that the enhanced Naive strategy possesses a (huge) bottleneck at the Querier level. This strategy is however economical as no verifications are performed. The Oracle strategy and DISPERS differ because of the setup (although the cost is relatively negligible) and mostly because of the verifications performed by the sampled targets ($2k$ instead of 1). This difference is inevitable as, in our context, it is impossible to trust a single signature but, thanks to the SEP2P protocol, we minimize it as much as possible (going from $2 \times (C + 1)$ to $2k$).

Additional Cost of the Cryptography and the Proxies

For information purposes, we bypassed the creation of the secure channel and the proxies and made measurements on the simulator, with DISPERS, in order to know the impact of these mechanisms on the total cost of an execution.

The results show that the additional costs is mainly "around" the targets, i.e. the encrypted

communications between the SR and BP and between the targets and the DA (via the AP). Thus, there are, at least, 4 asymmetric cryptographic operations per target. We must add up to these costs, the encrypted communications between the Querier and the actors and between the CI and the PS, SR, which can become important when the number of actors is too large (see Section 5.2.3).

Also note that the encrypted communications between the SR and the BP are only present if we use proxies. If not, the SR use the symmetric keys of the sampled targets (k_T) they recomposed (which we do not count). Hence, the additional cost of the proxies, in terms of cryptographic operations, is exactly 2 operations per target.

Influence of the Size of the Network (N)

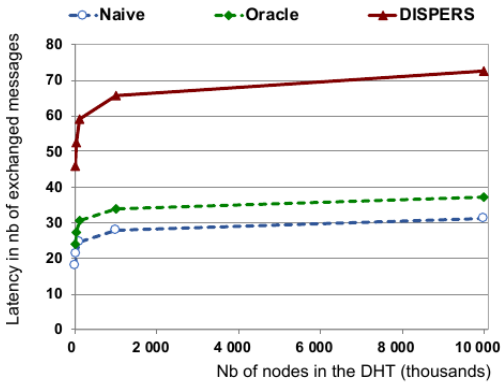

Figure 5.7: Comm. latency vs N

The size of the network does not have any impact on the cryptographic cost of any of the three strategies: the number of sampled targets and k stay the same. The impact on the latency and the total work in terms of number of communications is minor thanks to the DHT — a look up costs on average $\log_2(N)$ messages.

Influence of the Number of Actors (PS, SR, DA)

To simplify this analysis, we chose to have the same number of PS, SR than of DA. We then varied the number of PS, SR and DA between 4 and 128 in order to observe and understand the influence of that choice. We did not featured the enhanced Naive strategy as it does not have these actors. We only show the cryptographic latency and the total work in terms of cryptographic operations and number of messages — varying the number of actors has no impact on the latency in terms of number of messages.

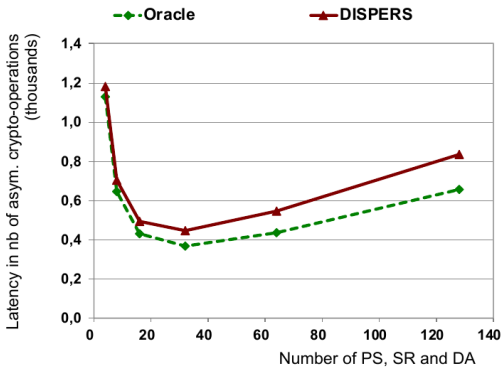

Figure 5.8: Crypto. latency vs nb of actors

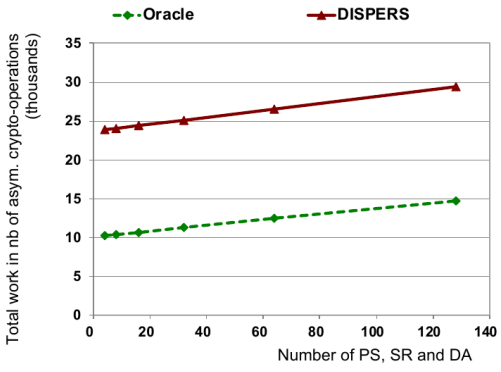

Figure 5.9: Crypto. total work vs nb of actors

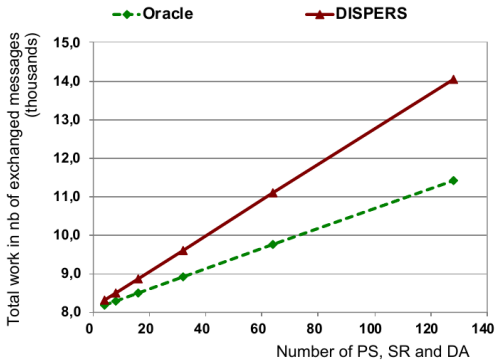

Figure 5.10: Comm. total work vs nb of actors

A first observation is that the cryptographic latency diminishes until a certain value, 32 for DISPERS, and increases again after (Figure 5.8). The reason is relatively simple and can be easily comprehended thanks to Figure 5.11 that gives, for each role, the number of cryptographic operations realized (note that the scale is logarithmic). Each series corresponds to an increasing number of actors. We observe that when the number of PS, SR increases, Q and the CI perform more cryptographic operations. Indeed, they have to encrypt their communications for more nodes. In contrast, the SR have less cryptographic operations to do as the number of targets is constant but they are more numerous. Likewise, the DA have to perform less decryption since they receive, on average, less messages. We can then easily understand that the latency diminishes as long as the gains at the SR and DA levels outweigh the additional costs imposed on Q and the CI. Hence, it could be interesting to have a number of DA slightly superior to the number of PS, SR.

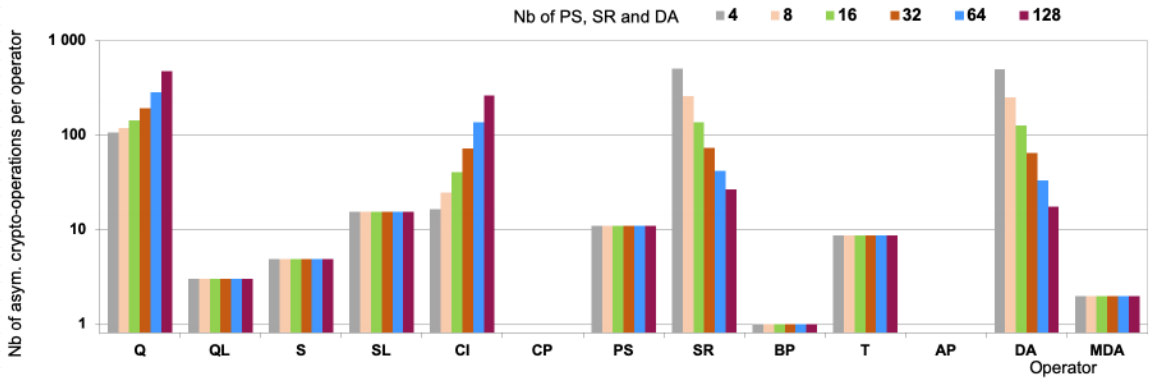

Figure 5.11: Nb of asymmetric crypto. operations per actor (varying PS, SR, DA)

We can draw similar conclusions based on the same graph portraying the number of exchanged messages.

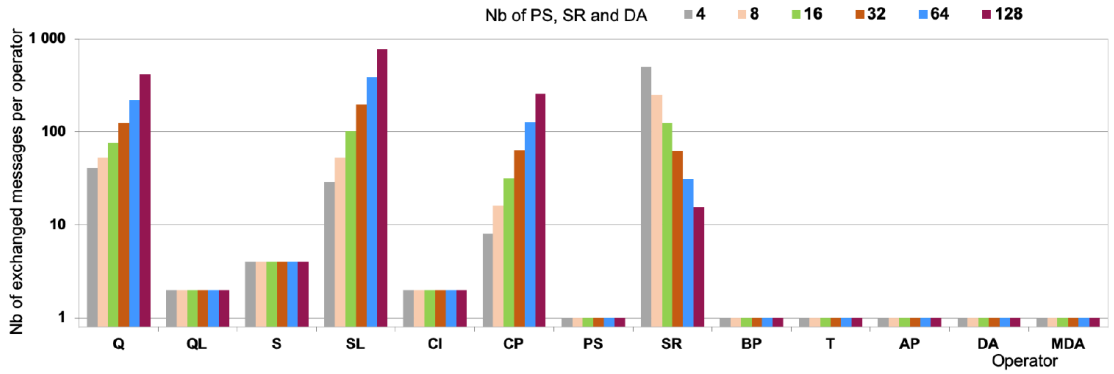

Figure 5.12: Nb of exchanged messages per actor (varying PS, SR, DA)

We observe that the number of messages for the SL is subject to the highest increase. This is explained by the fact that they have to ensure that the actors are online (through "pings"). Moreover, the CP also have more work as they are the one that have to communicate with the PS, SR even though the encryption was done by the CI.

Note that we do not notice any diminution of the number of messages at the DA level. This is only because this graph only takes into account the number of messages *sent*, and not received.

Hence, it could seem tempting to have a high number of actors (to notably diminish the impact of a corrupted actor), however increasing their numbers has an additional costs be it at the at the execution or the setup (also see Section 5.2.3). Furthermore, as discussed in the security analysis, increasing this number does not lessen the average leak (that stays on average at $c\%$).

Influence of the Number of Sampled Targets

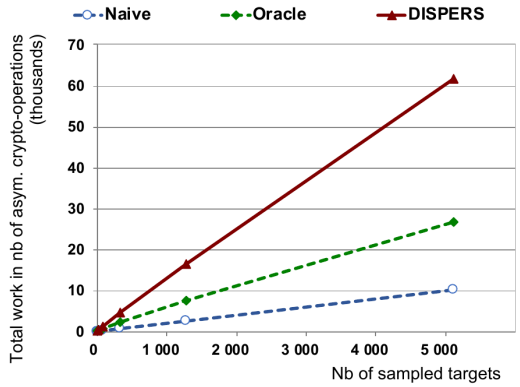

Figure 5.13: Crypto. tot. work vs nb of targets vs C

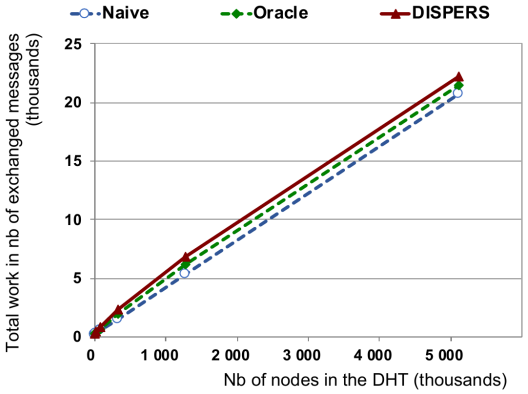

Figure 5.14: Comm. tot. work vs nb of targets

As expected given the previous conclusions we drew, the cost, in terms of number of messages or cryptographic operations, is linear with regard to the number of targets. We can observe a little flexion of the curves near the origin. This is due to a tweaking of the number of actors when there is not enough targets, in order to not have too much PS, SR or DA in comparison.

Influence of Shamir's Secret Sharing Scheme

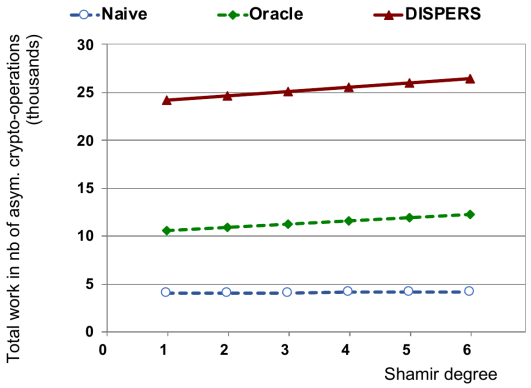

Figure 5.15: Crypto. total work w.r.t. Shamir's Scheme

We only show the curve regarding the total work in terms of cryptographic operations as the other curves are similar. As we can see, the impact of this parameter is minor. This is not surprising considering that the majority of the cost is concentrated around the targets. Additionally, as shown when we introduced this scheme (see Section 4.3.1), it is not necessary to set a high threshold value.

Setup versus Execution

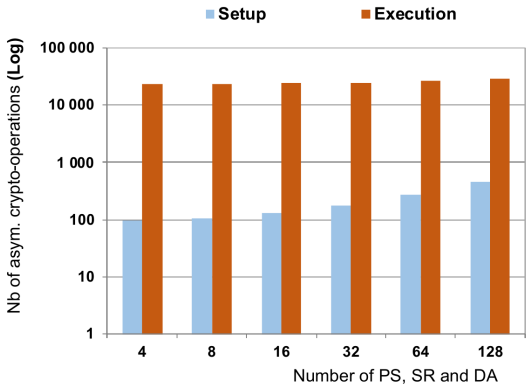

Figure 5.16: Crypto. total work, Setup vs Execution

With this final graph we can see that the setup cost is negligible (logarithmic scale) compared to the cost of executing the query, even with a large number of actors.

Conclusion

To conclude, we saw in this section the following elements: the enhanced Naive strategy is not suited to our context, even if we only consider the performances (the bottleneck at Querier is blocking). This forces a distribution of the knowledge and tasks. Moreover, this distribution, as illustrated by the Oracle strategy, has a relatively low cost and leads to under-linear leaks (in this regard the Oracle strategy is equivalent to DISPERS in Section 5.2.1). Unfortunately, the Oracle strategy is unrealistic, calling for a viable alternative able to produce a trustworthy list of actors.

At the same time, we saw that the setup cost induced by the SEP2P protocol is negligible compared to the execution cost. Plus, SEP2P helps drastically reducing the number of verifications performed.

Additionally, although DISPERS has to be configured, the influence of the parameters on the overall cost and latency is clear and enables us to set them appropriately.

Finally, we saw that DISPERS has a intrinsic cost that cannot be reduced: the targets must verify the list of actors.

5.3 DISPERS: Proof-of-Concept

In this section we discuss two proof-of-concept implementations of the DISPERS protocol. The first one is a web implementation for the purpose of showcasing its capabilities during the demonstration session of the VLDB conference held in 2019. The second is an implementation within the Cozy Cloud service as an experimental feature.

5.3.1 DISPERS Demonstration

The purpose of this demonstration is to illustrate the DISPERS system thanks to a simulator and a graphical front-end; and to demonstrate the rationale of three of our requirements:

task atomicity, *knowledge dispersion* and *imposed randomness*. We decided not to include the *hidden communications* requirements and the separation of the *TF* role (into pairs of (PS, SR)) for clarity — the lack of space and the complexity of the full DISPERS protocol making for a bad combination.

Also note that, at the time, we decided to split the list of actors into two distinct parts: the list of *TF* (*VTFL*) and the list of *DA* (*VDAL*). The rationale was to not disclose more information than needed to the actors. The introduction of the Merkle Hash Tree to link the list of actors to the query renders this separation impossible.

Other minor differences between the first screenshot and the following ones are due to improvements of the demonstration platform (in pursuit of a clearer and more explicit interface).

We introduce our approach using the graphical interface as depicted in Figure 5.17. Attendees can select or configure a query (top) and use the command panel (middle-right) to execute one of the query protocols, change the system parameters (e.g., number of colluding nodes), or run the protocol step by step. The last button allows exhibiting figures on the security and scalability of the DISPERS protocol.

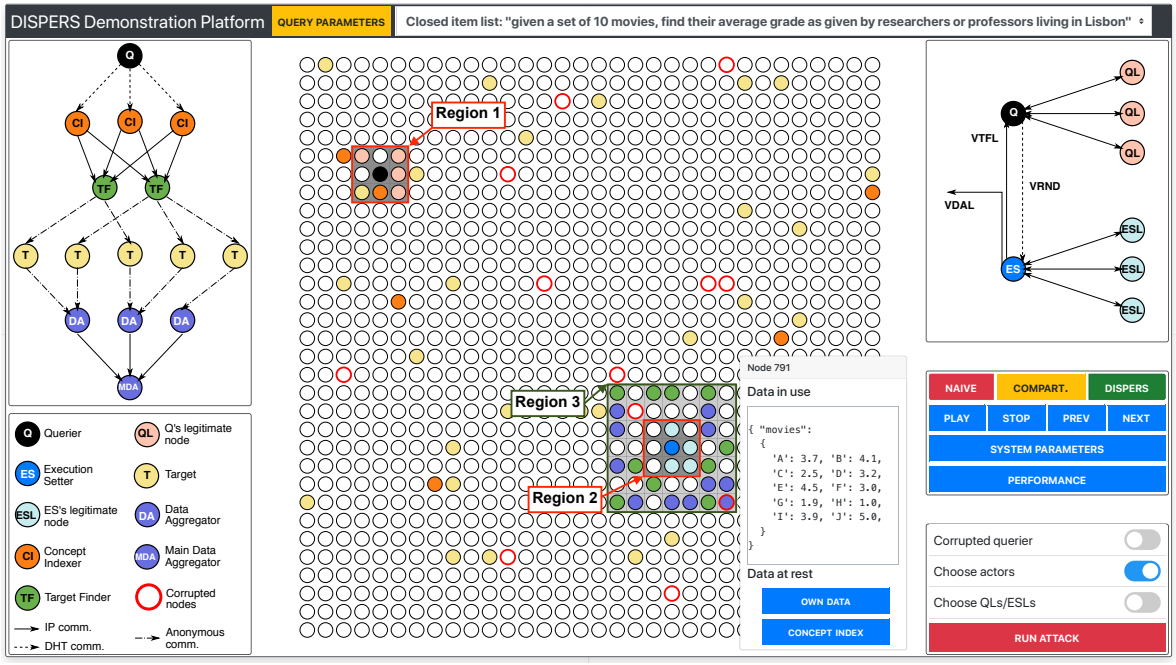

Figure 5.17: Demonstration platform

After explaining the demonstration platform, we focus on a given query and run protocols of increasing complexity and resistance to attacks (Naive, Compartmentalized and DISPERS), thus explaining the rationale of each requirement. Figures 5.18 to 5.21 illustrate the different steps of an execution with DISPERS.

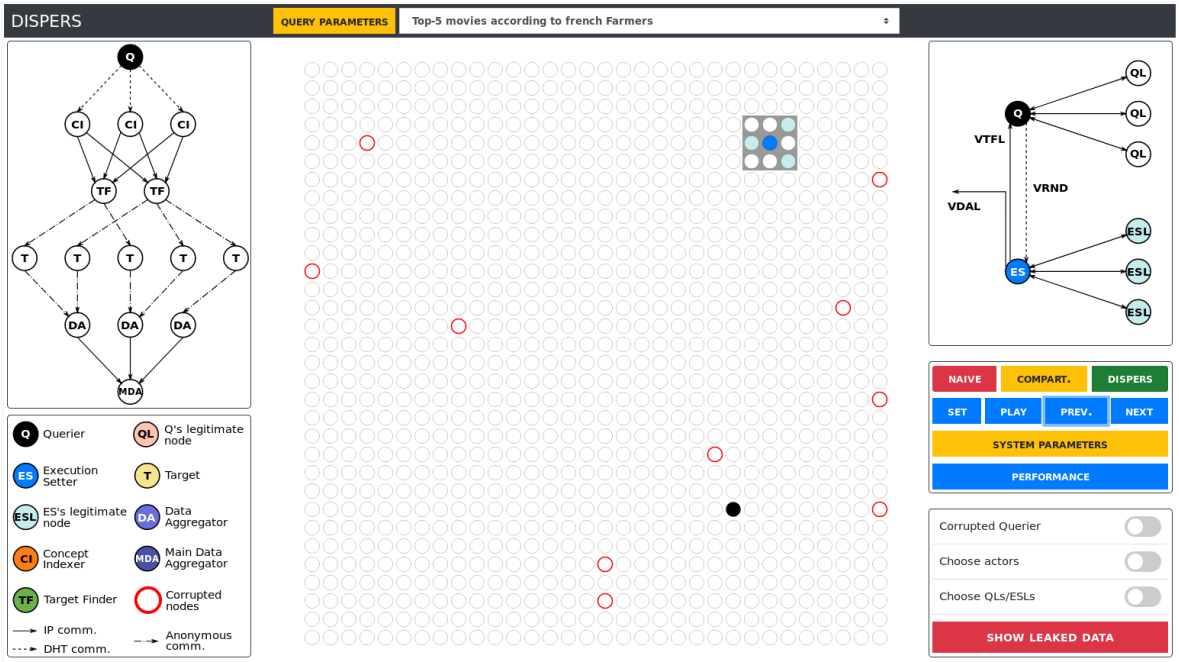

Figure 5.18: DISPERS Execution: *SL* nodes are highlighted.

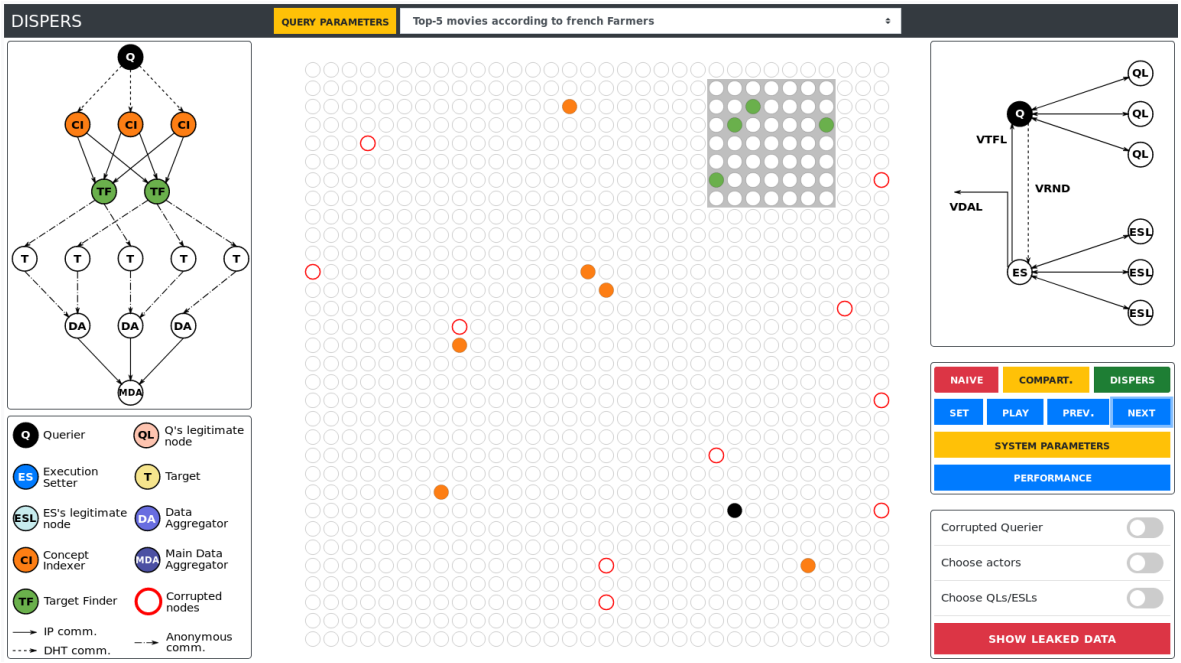

Figure 5.19: DISPERS Execution: *CI* and *TF* nodes are highlighted.

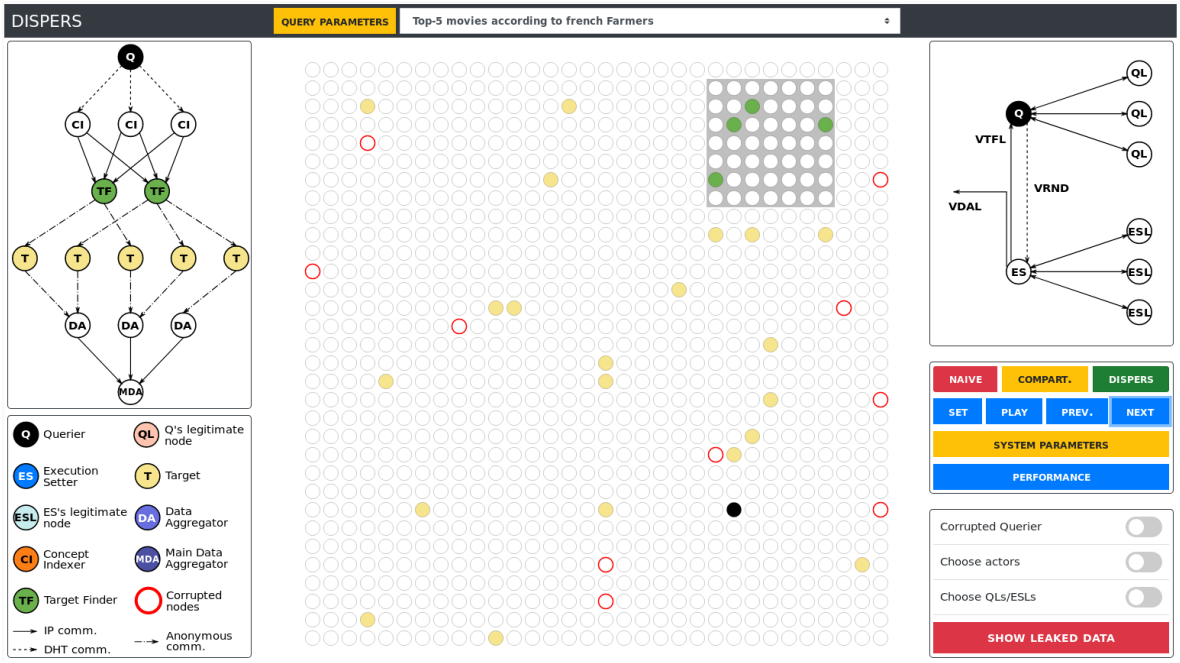

Figure 5.20: DISPERS Execution: *TF* nodes and the targets are highlighted.

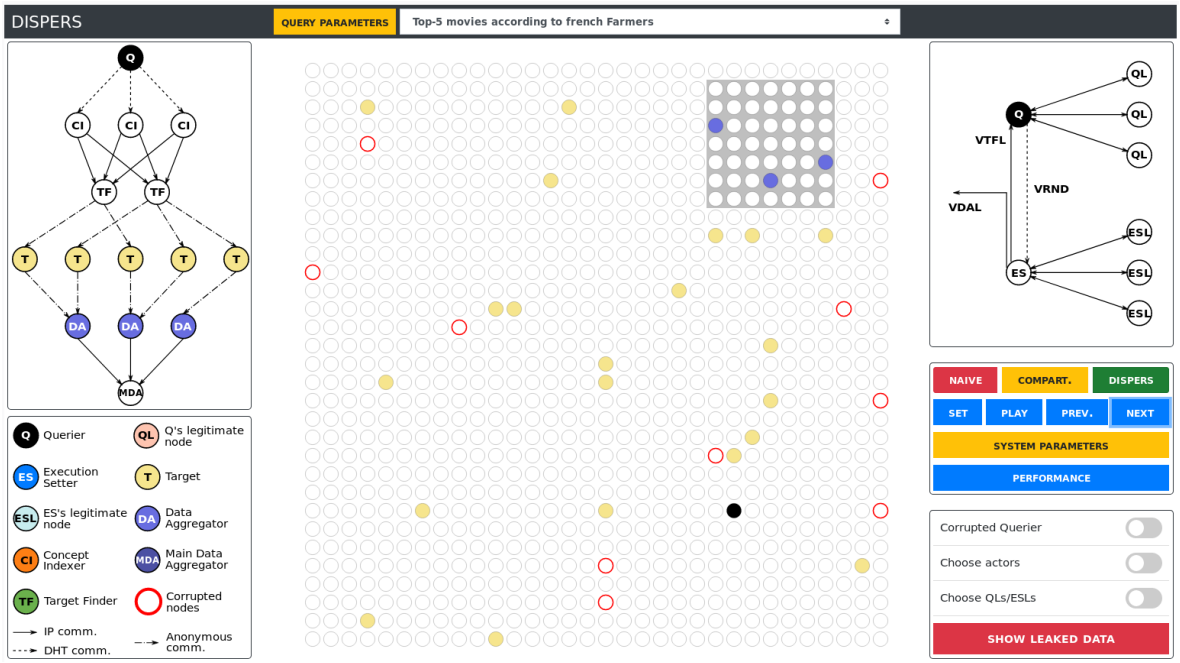

Figure 5.21: DISPERS Execution: the targets and *DA* nodes are highlighted.

Attendees can also try to hack the demonstration platform. The goal of this game is to achieve a deeper understanding of DISPERS by trying to defeat its security, e.g., exhibiting some confidential data of a given node. We expect them to defeat easily the naive protocol. Playing with the queries, the parameters and inspecting the content of colluding nodes, the attendee may also obtain the node's data with the compartmentalized protocol but we are

confident that this will be unfeasible with DISPERS.

5.3.2 Integration of DISPERS into Cozy Cloud

As of today and for the purposes of creating a solid user base (a commendable objective!), the Cozy Cloud architecture is almost completely centralized. Few users possess their own instances running on dedicated devices and the promise of equipping the PDMS nodes with Trusted Execution Environment is not fulfilled — although most of the devices possess it. In this context, implementing the DISPERS protocol and providing the same guarantees is then difficult and, as such, compromises are to be expected.

Nonetheless, a degraded version of the DISPERS protocol, in the context of the French ANR PerSoCloud, is currently being studied and an implementation — entirely realized by Cozy Cloud — under way. The PerSoCloud project is a joint effort between the French telecommunications corporation Orange, the Petrus research team (mixed from INRIA and the University of Versailles Saint-Quentin) and Cozy Cloud. Its objective is to provide sharing capabilities to the "Personal Cloud" paradigm — incarnated by the Cozy Cloud solution — on three dimensions: (i) *device sharing*, letting users share their data between various devices; (ii) *peer sharing*, letting users share their data with identified collaborating users; and (iii) *community-sharing*, letting users share personal data among a large community. The second objective of the project, of equal importance, is to ensure that these added capabilities come with strong privacy guarantees.

Integrating the DISPERS protocol to the Cozy Cloud service serves towards the fulfillment of the community-sharing capability by providing a framework to query the (logically) distributed data. To detail the ongoing work we first describe the system parameters and then give an overview of the altered protocol.

System Parameters and Threat Model

Given the context in which this study takes place, all the aspects of the DISPERS protocol can and have to be questioned. In the following we then take a look at the query, the actors, the threat model and the data model.

Query. The structure of a query remains identical: it is composed of a *target profile*, a *local query* and an *aggregate query*. Given the lack of distribution and hardware protection, the expressiveness of the profile and the possible computations are limited and the entire query is validated by a trusted third-party.

Actors. The first and most important actor of a query, is the Cozy Cloud infrastructure, referred to as the *stack*. The data of the users are stored in the stack. The stack possesses a pair of asymmetric cryptographic keys.

To distribute the computations, the stack is assisted by several *enclaves*. They are trustworthy external servers, potentially certified, that possess a Trusted Execution Environment. The different enclaves are isolated from each other and from the stack. They do not persist any personal data and possess a pair of asymmetric cryptographic keys.

Five different enclaves are required in order to launch a query:

1. an enclave *Querier (Q)*: the only entity able to launch a query and retrieve the result;
2. an enclave *Concept Indexer (CI)*: it is the only actor able to decipher the concepts;

3. an enclave *Target Finder (TF)*: it applies the pseudonymized target profile on the list of potential targets;
4. an enclave *Data (D)*: it deciphers the addresses of the targets and applies the local query on the data sent by the stack;
5. an enclave *Data Aggregator (DA)*: it deciphers the local results and applies the aggregate query (potentially pseudonymized).

Threat model. We assume the stack to be honest but curious: it can spy on the communications but respects the protocol and does not forge queries. We also aim at preventing a single corrupted enclave from disclosing the associations target $\Longleftrightarrow$ profile and target $\Longleftrightarrow$ local result. We do not consider a collusion of several enclaves.

Data model. Given the centralization of the data, the index in which the associations concept $\Longleftrightarrow$ addresses are stored is also centralized. To avoid disclosing those associations, the enclave *CI* generates as many symmetric keys as there are concepts and whenever a user wants to store a concept, she contacts (using a secure channel and from her device) the enclave *CI*, providing the concept, which then replies with the concept encrypted with the corresponding symmetric key. This encrypted concept acts as the key in the index, which prevents the stack from associating it to the user but still lets the stack generate the index. Similarly, to prevent the enclave *TF* from directly accessing the users' addresses, the users also contact the enclave *D* and ask for a symmetric key. They then encrypt their address with this key before sending both the encrypted concept and their encrypted address to the stack.

To prevent the stack from learning the local results, they also have to be stored encrypted. For that purpose, each user generates (directly on her device) a symmetric key and a unique pseudonym (e.g. a salted hash of this key). They then encrypt the data they agree to share with the symmetric key and this very same key plus the pseudonym with the public key of the enclave *DA*. They finally ask the stack to store the encrypted data and to send to the enclave *DA* the encrypted symmetric key and pseudonym. Proceeding that way lets the users choose which data can be queried and effectively prevents the stack from knowing which information is contributed. Additionally, asking the stack to transmit the encrypted pseudonym and symmetric key hides the user.

Note that to retrieve the encrypted data corresponding to a local query we suppose that some metadata information are still associated to the encrypted data and leveraged by the enclave *D* to filter out the relevant parts.

Cozy-DISPERS Protocol

The objective of this protocol is to enforce a first level of *task atomicity* and *knowledge dispersion* to reduce the impact of a leakage — the risk should theoretically be close to zero if the enclaves are indeed trustworthy.

Cozy-DISPERS protocol

---

(0) The query is validated by a trustworthy third-party authority.
(1) The enclave *Q* generates a pseudonym for each concept of the target profile. It sends the associations pseudonym $\Longleftrightarrow$ concept to the enclave *CI*. In parallel, it sends the pseudonymized target profile to the enclave *TF*, the local query to the enclave *D* and the pseudonymized aggregate query to the enclave *DA*.

(2) The enclave *CI* encrypts the concepts with their matching symmetric key and asks the stack to send the list of associations corresponding to the encrypted concepts to the enclave *TF* and to replace the encrypted concepts with their pseudonyms.
(3) The stack retrieves the lists associated to the different encrypted concepts, and sends them to the enclave *TF* — with the pseudonym in lieu of the encrypted concept.
(4) The enclave *TF* applies the pseudonymized target profile on the lists and sends the (sampled) encrypted targets' addresses to the enclave *D*.
(5) The enclave *D* deciphers the (sampled) targets' addresses and asks the stack for the encrypted data of these targets. Upon reception of the encrypted data, the enclave *D* extracts the relevant part by applying the local query, and send the encrypted local results to the enclave *DA*.
(6) The enclave *DA* deciphers the local results, applies the pseudonymized aggregate query and sends the final result to the enclave *Q*.

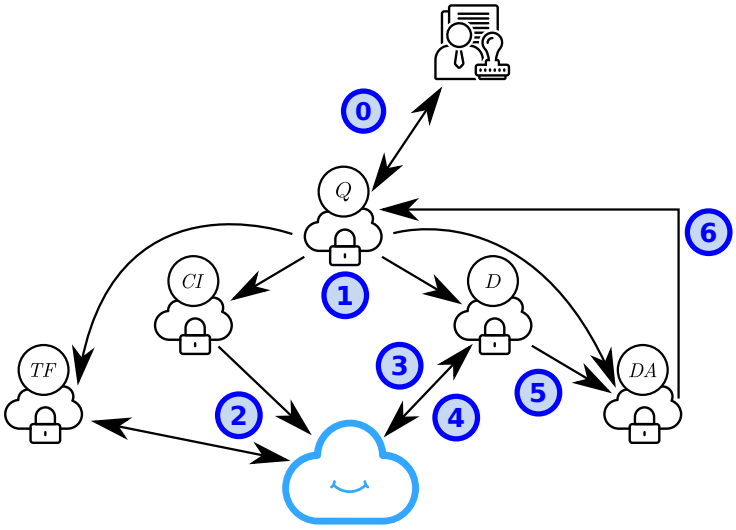

Figure 5.22: Cozy-DISPERS protocol

Let us study the impact of a corrupted enclave under this protocol:

- If the enclave *Q* is corrupted it learns the final result and the query parameters.
- If the enclave *CI* is corrupted it only has access to the concepts and, during an execution, to the associated pseudonyms.
- If the enclave *TF* is corrupted then no sensitive information is leaked: it only has access to encrypted addresses and to the pseudonymized target profile.
- If the enclave *D* is corrupted then the addresses of the targets are leaked but it cannot tie them to a profile or to some local results.
- If the enclave *DA* is corrupted it can access the local results without being able to tie them to a profile or to any target.

Moreover, the stack also sees very little information: it only knows which users are targeted by a query as it provides their encrypted data to the enclave *D*. Since it does not know the target profile or which data are extracted by the local query it cannot infer anything.

Hence, under the considered, system parameters this first implementation achieves the objectives set in the PerSoCloud project: users can share personal data among a large community and securely query them.

5.4 Conclusion

We started this chapter with a security analysis of DISPERS in which we saw that SEP2P, the setup phase of DISPERS, leads to a linear leakage as there is, on average, the same proportion of colluding nodes in the random list of actor than in the network. We then saw that by correctly distributing the tasks and the knowledge, DISPERS renders this linear leakage under-linear: to obtain information an attacker has to have certain combinations of actors, the associated probability being c^2. Furthermore, enforcing a query budget, sampling the targets and hiding the targets with proxies severely limits the leeway of an attacker: it cannot repeatedly query the network in hope of a favorable list of actors, it cannot uncover the entire list of targets and it cannot associate a result to a target.

We then evaluated the security and performance of DISPERS. In accordance with our analysis, the security evaluation shows that all the security measures are mandatory to reach an under-linear leakage. The performance additionally showed that DISPERS is the only viable solution as, even when enhanced, a Naive strategy is inappropriate because of the Querier acting as a single point of failure and a major bottleneck. We then compared DISPERS to an Oracle strategy providing a random list of actors to assess the impact of the different parameters. This comparison shows that the distribution of the tasks and knowledge has a minor impact on the overall performances, that the influence of the parameters is relatively clear which argues for an easy tweaking, and finally that the only irreducible cost of DISPERS is the verifications performed by the targets — which we already minimize via SEP2P.

The last section described two proof-of-concepts of DISPERS. The first takes the form of a demonstration during which we presented an interactive graphical user interface where attendees were invited to try to hack the system in order to uncover knowledge regarding a specific node. The main purpose being to demonstrate the capabilities of our solution. The second takes the form of a degraded implementation of the DISPERS protocol in the Cozy Cloud service, in order to provide a *community-sharing* feature in the context of the PerSoCloud project. This implementation relies on external trusted servers to provide the task and knowledge separations.

Chapter 6

Conclusion

6.1 Summary of the Contributions

Privacy is a growing concern in today's society. The constant spying by few corporations and the centralization that ensues can lead to drastic losses in terms of security and control of our own lives. Hence, we have to propose viable alternatives: decentralized, privacy focused solutions that are on par feature-wise with their centralized counterparts. The Personal Cloud paradigm follows this trend and proposes a digital space, the equivalent of a *digital home*, where users can regain control of their data. Building on this paradigm, this thesis proposes to assemble a fully-distributed, secure and privacy-preserving query processing system on top of a network of Personal Clouds. In particular, the different contributions are as follows:

1. We propose a set of four requirements detailing the specific features any solution, evolving in the same environment as the one we describe, should respect: *imposed randomness* — preventing an attacker from influencing the selection of the actors in the execution; *knowledge dispersion* — preventing any node from concentrating information it does not own; *task atomicity* — splitting the execution in as many independent tasks as necessary so as to minimize the sensitive information each actor has access to; and *hidden communications* — protect the identity of the sensitive nodes as well as the content of their communications, to prevent an attacker spying on the network from intercepting sensitive data.

2. We propose *SEP2P*, a protocol that leverages the Distributed Hash Table overlay organizing the network and the CSAR protocol to enforce the *imposed randomness* requirement and generate a verifiable random list of actors. This protocol's main advantage resides in its ability to only require the participation of a limited set of nodes (extremely small in comparison to the number of colluding nodes) to achieve its objective: by imposing the location of nodes in the DHT we are able to take localized decisions and attain a very high probability of involving at least one honest node in the generation of the list — which is our only condition to generate one.

3. We propose *DISPERS*, a protocol that applies the last three requirements to split and distribute the execution of a query to a set of randomly selected actors. We define a query as the combination of a *target profile*, a *local query* and an *aggregate query*. More especially, the target profile has the dual objective of restraining the participants by

specifying relevance criteria. We supplement this selection by a *sampling* that not only enforces this limitation but does so while preserving, to some extent, the quality of the result and actually limiting the impact of a leakage. To be able to retrieve nodes based on the profile, *concepts* are stored in the DHT. As each concept can be sensitive we split them in shares following Shamir's Secret Sharing Scheme.

We then successively apply the requirements which leads to the definition of three distinct roles: *Profile Sampler* — which apply the target profile and operate a sampling; *Share Recomposer* — which reconstruct the IP addresses of the sampled targets; and *Data Aggregator* — which computes the aggregate query on the data sent by the sampled targets.

These roles are complemented by proxies in order to hide, and protect, the identity of the targets. Additional protections are put in place to severely limit attack scenarios based on extensive generation of actors' lists or query executions: a *query budget* limits the number of queries and a *Merkle Hash Tree* coupled with the signatures of the list builders links the list of actors to a specific query.

4. We finally propose an in-depth security analysis and performance evaluation of the DISPERS protocol, as well as two proof-of-concepts. We show in the security analysis that, although SEP2P selects a linear amount of colluding nodes among the actors (equal, on average, to the proportion of colluding nodes in the entire network), the different security mechanisms employed in DISPERS leads to an under-linear leakage: the optimal leakage we could hope for given our assumptions! This fact is due to separation of concerns and knowledge between the actors that makes an attacker only gain knowledge when several roles are simultaneously played by colluding nodes.

   The first proof-of-concept is a hackable graphical interface built for a demonstration session during which attendees can try to bypass the security mechanisms present in DISPERS, in order to better grasp our contribution. The second proof-of-concept takes the form of a degraded implementation of DISPERS for the French ANR PerSoCloud project. By adapting the requirements to a centralized context and bringing in secure trustworthy external servers, called *enclaves*, we achieve a similar separation of concerns than in DISPERS and bring community sharing capabilities to the Cozy Cloud service.

6.2 Future Work

Given the constant digitalization of our lives and being, to the best of our knowledge, the first work tackling the issue of querying a fully-distributed network of Personal Clouds in a privacy-preserving manner, there are many research directions that could be developed following it. For instance, how to further protect the result of a query: it would be particularly interesting to see how can we combine Privacy-Preserving Data Publishing techniques with our protocol. Also, investigating optimizations for the aggregation phase of our protocol yields interesting perspectives: although our approach is generic, specific computations could benefit from specific approaches.

Nevertheless, we believe the following research directions to be the most promising:

- **Understanding the duality that exists between integrity and confidentiality.** In this work we provided a way to preserve the confidentiality of the users and of their

data: not knowing which node is at the origin of which data, not knowing the profile associated to a node, and, when applicable, not knowing the nature of the data manipulated. In other words, confidentiality means knowing as little as possible.

Conversely, integrity deals with being and making sure that nothing was *altered* during a process: verifications are thus required. For instance, providing a hash and a signature of a piece of information is an inexpensive and reliable way of checking that it was not modified. But, in general[1], to perform these verifications the original data is required.

Hence, in our context, providing confidentiality and integrity seems to involve a major trade-off: either nodes know as little as possible to protect their confidentiality, at the risk of losing precision or leaving some leeway for malicious nodes to slightly alter the query; or either we cannot guarantee a complete confidentiality but we guarantee a correct execution. An example of this trade-off in our protocol is the selection of targets: as a target cannot verify if her profile matches the target profile (as she cannot access the profile as it would disclose it to, at least, the *Share Recomposer*), a malicious node could theoretically include other, "unwanted", nodes as targets.

Therefore, better understanding the duality between integrity and confidentiality could lead to the creation of alternative protocols that could focus on offering specific protections.

- **Machine learning applications.** Machine learning is an extremely active and prolific research area. Industrials are also highly interested in this subject with new projects and frameworks coming out regularly (Tensorflow, autonomous cars, automatic face detection).

  To train a Machine learning algorithm large amount of data are needed. More particularly these data-sets must respect three criteria: they should be *big* (as in Big Data), they should be *diverse* and they should be *real* (as in real life data). Putting together such a massive database poses several serious problems: first and foremost preserving the privacy of the (willing or unwilling) participants, providing a way for the users to easily contribute, and keeping the data as close as possible to its "real" value as anonymization techniques lessens the quality of the data-set which then hinders the efficiency of the training.

  An architecture à la DISPERS appears to solve all these issues: the Personal Cloud offers the perfect medium to ask for the consent of the users and through DISPERS privacy-preserving query system it is possible to extract the relevant real data from a large number of willing participating nodes — without revealing their identity, aggregate them so as to render them compatible with the training process and finally train the algorithm. However, decomposing a machine learning training phase in atomic tasks (a requirement to provide a privacy-preserving execution) is an open and intricate question: for example, training an algorithm usually assumes a lot of back and forth which raises many confidentiality issues.

[1] We do not consider Zero-knowledge based solutions as they mostly rely on cryptography which, as stated in Chapter 2, are not compatible with our environment.

Appendix A

Detail of the information accessed by each Actor

In the following we detail for each actor, the information it receives — the *input* — and the information it produces — the *output*.

For the input we detail from whose actor the information came, if it was sent encrypted, if so with which encryption key and if a signature is checked upon reception.

For the output we detail to whom it will be sent, if it is encrypted, if so with which key and if the node computed a signature to attest it.

A.1 Querier Q

Input

Sender	**Information**	**Encrypted?**	**Key**	**Decryption?**	**Verification?**
S	VAL	×	—	No	No

Output

Recipient	**Information**	**Encryption?**	**Key**	**Signature?**
PS	QID tp° $salt_{tp^{\circ}}$ $MHT_{/PS}$	Yes	$kpub_{PS}$	No

Recipient	Information	Encryption?	Key	Signature?
SR	VAL lq $salt_{lq}$ $MHT_{/T}$	Yes	$kpub_{SR}$	No

Recipient	Information	Encryption?	Key	Signature?
DA	QID aq $salt_{aq}$ $MHT_{/DA}$	Yes	$kpub_{DA}$	No

Recipient	Information	Encryption?	Key	Signature?
MDA	QID aq $salt_{aq}$ $MHT_{/DA}$ $cert_Q$	Yes	$kpub_{DA}$	No

Recipient	Information	Encryption?	Key	Signature?
CI	VAL c $salt_c$ c° $MHT_{/CI}$	Yes	$kpub_{CI}$	No

A.2 Concept Indexor *CI*

Input

-

Sender	Information	Encrypted?	Key	Decryption?	Verification?
Q	VAL	✓	$kpub_{CI}$	Yes	Yes
	c	✓			
	$salt_c$	✓			
	c°	✓			
	$MHT_{/CI}$	✓			

Output

-

Recipient	Information	Encryption?	Key	Signature?
CP $(\rightarrow PS)$	QID	Yes	$kpub_{PS}$	No
	c°			
	IP_{SR}			
	$kpub_{SR}$			
	TS_Q			
	$\ulcorner MHT \urcorner$			
	$\left\{ \begin{array}{c} CTID \\ TID_{share} \\ [k_T IP_{share}]^{k_{tmp}} \end{array} \right\}$	Yes	k_{tmp}	
	IP_{PS}	No	—	

-

Recipient	Information	Encryption?	Key	Signature?
CP $(\rightarrow SR)$	QID	Yes	$kpub_{SR}$	No
	$\left\{ \begin{array}{c} k_{tmp} \\ hash([k_T IP_{share}]^{k_{tmp}}) \end{array} \right\}$			
	IP_{SR}	No	—	

A.3 Concept indexor Proxy *CP*

Input

- *CI* ($\rightarrow$ *PS*)

Sender	Information	Encrypted?	Key	Decryption?	Verification?
CI ($\rightarrow$ *PS*)	QID	✓	$kpub_{PS}$	No	No
	c°	✓			
	IP_{SR}	✓			
	$kpub_{SR}$	✓			
	TS_Q	✓			
	$\ulcorner MHT \urcorner$	✓			
	$\{CTID, TID_{share}, [k_T IP_{share}]^{k_{tmp}}\}$	✓			
		✓	k_{tmp}		
	IP_{PS}	×	—		

- *CI* ($\rightarrow$ *SR*)

Sender	Information	Encrypted?	Key	Decryption?	Verification?
CI ($\rightarrow$ *SR*)	QID	✓	$kpub_{SR}$	No	No
	$\{k_{tmp}, hash([k_T IP_{share}]^{k_{tmp}})\}$	✓			
		✓			
	IP_{SR}	×	—		

Output

- *PS*

Recipient	Information	Encryption?	Key	Signature?
PS	QID	Yes	$kpub_{PS}$	No
	c°			
	IP_{SR}			
	$kpub_{SR}$			
	TS_Q			
	$\ulcorner MHT \urcorner$			
	$\{CTID, TID_{share}, [k_T IP_{share}]^{k_{tmp}}\}$			
			k_{tmp}	

Recipient	Information	Encryption?	Key	Signature?
SR	QID $\{k_{tmp}$ $hash([k_T IP_{share}]^{k_{tmp}})\}$	Yes	$kpub_{SR}$	No

A.4 Profile Sampler *PS*

Input

Sender	Information	Encrypted?	Key	Decryption?	Verification?
Q	QID	✓	$kpub_{PS}$	Yes	Yes
	tp°	✓			
	$salt_{tp^\circ}$	✓			
	$MHT_{/PS}$	✓			
CP $(CI \rightarrow)$	QID	✓	$kpub_{PS}$	Yes	No
	c°	✓			
	IP_{SR}	✓			Yes
	$kpub_{SR}$	✓			
	$\ulcorner MHT \urcorner$	✓			
	$\{CTID$ TID_{share} $[k_T IP_{share}]^{k_{tmp}}\}$	✓ ✓			No
		✓	k_{tmp}	No	

Output

Recipient	Information	Encryption?	Key	Signature?
SR	QID	Yes	$kpub_{SR}$	No
	$\{[k_T IP_{share}]^{k_{tmp}}\}$	Yes	k_{tmp}	

A.5 Share Recomposer SR

Input

-

Sender	Information	Encrypted?	Key	Decryption?	Verification?
Q	VAL	✓	$kpub_{SR}$	Yes	No
	lq	✓			
	$salt_{lq}$	✓			
	$MHT_{/T}$	✓			
CP $(CI \rightarrow)$	QID	✓	$kpub_{SR}$	Yes	No
	$\left\{ k_{tmp} \right.$	✓			
	$\left. hash([k_T IP_{share}]^{k_{tmp}}) \right\}$	✓			
PS	QID	✓	$kpub_{SR}$	Yes	No
	$\{[k_T IP_{share}]^{k_{tmp}}\}$	✓	k_{tmp}	Yes	

Output

-

Recipient	Information	Encryption?	Key	Signature?
BP	VAL	Yes	k_T	No
	lq			
	$salt_{lq}$			
	$MHT_{/T}$			
	$selector(k_T)$	Yes	$kpub_{BP}$	
	IP_T			

A.6 Before Proxy BP

Input

-

Sender	Information	Encrypted?	Key	Decryption?	Verification?
SR	VAL	✓	k_T	No	No
	lq	✓			
	$salt_{lq}$	✓			
	$MHT_{/T}$	✓			
	$selector(k_T)$	✓	$kpub_{BP}$	Yes	
	IP_T	✓			

Output

Recipient	Information	Encryption?	Key	Signature?
T	VAL lq $salt_{lq}$ $MHT_{/T}$	Yes	k_T	No
	$selector(k_T)$	No	—	

A.7 Target T

Input

Sender	Information	Encrypted?	Key	Decryption?	Verification?
BP $(SR \rightarrow)$	VAL lq $salt_{lq}$ $MHT_{/T}$	✓ ✓ ✓ ✓	k_T	Yes	Yes
	$selector(k_T)$	×	—	No	No

Output

Recipient	Information	Encryption?	Key	Signature?
AP $(\rightarrow DA)$	QID TS_Q $\ulcorner MHT \urcorner$ IP_{MDA} $kpub_{MDA}$ res_T	Yes	$kpub_{DA}$	No
	IP_{DA}	No	—	

A.8 After Proxy AP

Input

- T ($\rightarrow DA$)

Sender	Information	Encrypted?	Key	Decryption?	Verification?
T ($\rightarrow DA$)	QID	✓	$kpub_{DA}$	No	No
	TS_Q	✓			
	$\ulcorner MHT \urcorner$	✓			
	IP_{MDA}	✓			
	$kpub_{MDA}$	✓			
	res_T	✓			
	IP_{DA}	×	—		

Output

- DA

Recipient	Information	Encryption?	Key	Signature?
DA	QID	Yes	$kpub_{DA}$	No
	TS_Q			
	$\ulcorner MHT \urcorner$			
	IP_{MDA}			
	$kpub_{MDA}$			
	res_T			

A.9 Data Aggregator *DA*

Input

-

Sender	Information	Encrypted?	Key	Decryption?	Verification?
Q	QID	✓	$kpub_{DA}$	Yes	No
	aq	✓			Yes
	$salt_{aq}$	✓			
	$MHT_{/DA}$	✓			
AP $(T \rightarrow)$	QID	✓	$kpub_{DA}$	Yes	No
	$\ulcorner MHT \urcorner$	✓			Yes
	TS_Q	✓			
	IP_{MDA}	✓			
	$kpub_{MDA}$	✓			
	res_T	✓			No

Output

-

Recipient	Information	Encryption?	Key	Signature?
MDA	QID	Yes	$kpub_{MDA}$	No
	$\ulcorner MHT \urcorner$			
	TS_Q			
	res_{DA}			

A.10 Main Data Aggregator *MDA*

Input

-

Sender	Information	Encrypted?	Key	Decryption?	Verification?
Q	QID	✓	$kpub_{MDA}$	Yes	No
	aq	✓			Yes
	$salt_{aq}$	✓			
	$MHT_{/DA}$	✓			
	$cert_Q$	✓			No
DA	QID	✓	$kpub_{MDA}$	Yes	No
	$\ulcorner MHT \urcorner$	✓			Yes
	TS_Q	✓			
	res_{DA}	✓			

Output

-

Recipient	Information	Encryption?	Key	Signature?
Q	res	Yes	$kpub_Q$	No

Annexe B

Résumé en Français du manuscrit

L'évolution des technologies fait que de plus en plus de tâches de notre vie de tous les jours sont automatisées. D'un point de vue « données », cette automatisation se traduit en une génération croissante *d'informations personnelles structurées* — c'est-à-dire facilement exploitables par un ordinateur et centrées sur nous.

Bien qu'elle permette une meilleure intégration de ces technologies, cette situation ne soulève pas moins des inquiètudes, notamment concernant notre vie privée : la plupart des interactions que nous avons avec ces technologies sont centralisées par les mêmes acteurs, ce qui leur donne ainsi une vue très détaillée de ce que nous faisons et donc de qui nous sommes.

Heureusement tout n'est pas noir dans le paysage numérique actuel : l'évènement récent le plus marquant a été l'adoption par l'Union Européenne du Règlement Général sur la Protection des Données. Ce règlement oblige notamment les acteurs manipulant des données à caractère personnel à définir explicitement comment celles-ci sont utilisées, à demander le consentement des utilisateurs avant toute collecte et qui offre à ces derniers la possibilité de demander une copie de leurs données.

Combiné avec l'initiative du *Nuage Personnel* ("Personal Cloud" en anglais), qui fournit aux utilisateurs un espace numérique dédié et sous leur seul contrôle — comme ce que propose la jeune pousse française Cozy Cloud, entreprise avec laquelle ces travaux ont été réalisés — il devient possible de construire une plateforme respectueuse de la vie privée.

Cependant, cette plateforme reste mono-utilisateur et, sans un cadre adéquat, ne permet pas de créer des applications multi-utilisateurs offrant les mêmes garanties. L'objectif de cette thèse est donc de définir un tel cadre pour permettre, plus spécifiquement, l'exécution de requêtes distribuées respectueuses de la vie privée sur un ensemble de Nuages Personnels.

Ainsi, pour répondre à la problématique exposée, nous faisons les contributions suivantes :

1. Dans le chapitre 2, nous commençons par établir les briques de bases de notre solution. En effet, ces travaux sont à l'intersection de trois domaines de recherche : les *Systèmes de Gestion de Données Personnelles* (dans lequel s'inscrit le Nuage Personnel), les *Techniques de Préservation de la Vie Privée* et les *Systèmes Distribués*. Or, chacun de ces domaines ne traite qu'un sous-ensemble des contraintes que nous avons : les Systèmes de Gestion de Données Personnelles n'ont pas tous le même modèle de confiance et/ou fournissent des fonctionnalités différentes ; les Techniques de Préservation de la Vie Privée peuvent être divisées en deux catégories, celles basées sur du chiffrement et celles qui font de la « publication de données respectueuse de la vie privée » (*Privacy-Preserving Data Publishing*),

qui ont chacune leurs avantages et inconvénients ; et les Systèmes Distribués répondent principalement aux problèmes d'efficacité et de disponibilité dans un réseau pair-à-pair.

2. En s'appuyant sur les technologies identifiées nous définissons dans le chapitre 3 un ensemble de quatre contraintes que n'importe quelle solution évoluant dans le même environnement devrait respecter : (i) l'*aléa imposé* — pour empêcher un attaquant d'influencer la sélection des acteurs lors de l'exécution d'une requête ; (ii) la *dispersion des connaissances* — pour qu'aucun participant ne concentre trop d'informations ; (iii) les *tâches atomiques* — diviser l'exécution d'une requête en un ensemble de sous-tâches de sorte à limiter au strict minimum les informations accédées par chaque acteur ; et (iv) les *communications « cachées »* — afin de protéger l'identité des participants ainsi que le contenu des communications.

3. Toujours dans le chapitre 3, nous proposons le protocole `SEP2P` qui utilise la Table de Hachage Distribuée organisant le réseau de Nuages Personnels pour faire respecter la première contrainte, l'*aléa imposé*, et générer une liste aléatoire et vérifiable d'acteurs. L'avantage principal de ce protocole réside dans sa capacité à ne nécessiter qu'un nombre restreint de participants (extrêmement faible par rapport au nombre de participants corrompus) pour atteindre cet objectif : en imposant la localisation des Nuages Personnels dans le réseau, nous sommes en mesure de prendre des décisions localisées tout en maintenant une très forte probabilité d'impliquer au moins un participant honnête dans la génération de la liste — ce qui est notre unique condition pour en produire une qui soit à la fois aléatoire et vérifiable.

4. Nous proposons, dans le chapitre 4, le protocole `DISPERS` qui, pour respecter les trois dernières contraintes, distribue l'exécution d'une requête entre différents acteurs (choisis aléatoirement). Nous définissons une requête comme la combinaison d'un *profil cible*, d'une *requête locale* et d'une *requête d'agrégat*. Plus particulièrement, le profil cible a le double objectif de restreindre le nombre de participants en ne sélectionnant que ceux qui sont pertinents pour la requête considérée. Pour réaliser cette sélection basée sur un profil, des *concepts* sont conservés dans la Table de Hachage Distribuée — une combinaison logique de concepts constituant un profil. Ces concepts pouvant être sensibles, nous les protégeons en les chiffrant en utilisant la *Technique du Partage de Secret de Shamir*.

   En appliquant successivement les différentes contraintes nous définissons les trois types d'acteurs suivants : *Profile Sampler* — qui appliquent le profil cible et effectuent un échantillonnage ; *Share Recomposer* — qui reconstruisent les adresses IP des participants sélectionnés ; et *Data Aggregator* — qui calculent la requête d'agrégat sur les données envoyées par les participants sélectionnés.

   Ces rôles sont complétés par des proxy qui cachent et protègent l'identité des participants sélectionnés. Des protections supplémentaires sont également mises en place pour sévèrement limiter des scénarios d'attaques qui reposent sur une génération excessive de listes d'acteurs ou d'exécutions de requêtes : un *budget* limite le nombre de requêtes qu'un participant peut faire et un *Arbre de Merkle* couplé avec les signatures des participants qui ont généré la liste des acteurs lie une liste d'acteurs à une requête spécifique.

5. Enfin, dans le chapitre 5, nous proposons une analyse détaillée de la sécurité et des performances de `DISPERS` ainsi que deux implémentations de type « preuve de concept ». Nous montrons notamment dans l'analyse de sécurité que, bien que `SEP2P` sélectionne une quantité linéaire de participants corrompus parmi les acteurs (égale, en moyenne, à la proportion

de participants corrompus dans tout le réseau), les différents mécanismes de sécurité employés dans DISPERS conduisent à une fuite sous-linéaire : la fuite minimale qu'on pouvait espérer étant donné nos hypothèses.

La première preuve de concept est une interface graphique réalisée dans le cadre d'une session de démonstration où les personnes présentes pouvaient tenter de contourner les mécanismes de sécurité mis en place (afin de mieux les comprendre). La seconde preuve de concept prend la forme d'une implémentation dégradée de DISPERS pour les besoins du projet ANR PerSoCloud. En adaptant les contraintes à un contexte centralisé et en s'appuyant sur des serveurs tiers de confiance, appelés *enclaves*, nous pouvons mettre en place une séparation similaire des tâches et donc des connaissances.

Bibliography

[1] Michael Abd-El-Malek et al. "Fault-scalable Byzantine fault-tolerant services". In: *SOSP*. ACM, 2005, pp. 59–74.

[2] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". In: *ACM Comput. Surv.* 51.4 (2018), 79:1–79:35.

[3] Mahdi Nasrullah Al-Ameen and Matthew K. Wright. "Design and evaluation of persea, a sybil-resistant DHT". In: *AsiaCCS*. ACM, 2014, pp. 75–86.

[4] Tristan Allard et al. "Secure personal data servers: a vision paper". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 25–35.

[5] Ittai Anati et al. *Innovative Technology for CPU Based Attestation and Sealing*. Tech. rep. intel, 2013, p. 7.

[6] Nicolas Anciaux et al. "Personal Data Management Systems: The security and functionality standpoint". In: *Information Systems* (2018).

[7] Nicolas Anciaux et al. "Trusted Cells: A Sea Change for Personal Data Services". In: *CIDR*. www.cidrdb.org, 2013.

[8] Android. *Android Key Store*. 2019. URL: `https://developer.android.com/training/articles/keystore.html`.

[9] Pierre-Louis Aublin et al. "The Next 700 BFT Protocols". In: *ACM Trans. Comput. Syst.* 32.4 (2015), 12:1–12:45.

[10] Yonatan Aumann and Yehuda Lindell. "Security against covert adversaries: Efficient protocols for realistic adversaries". In: *Theory of Cryptography Conference*. Springer. 2007, pp. 137–156.

[11] Baruch Awerbuch and Christian Scheideler. "Towards a scalable and robust DHT". In: *SPAA*. ACM, 2006, pp. 318–327.

[12] Michael Backes et al. "CSAR: A Practical and Provable Technique to Make Randomized Systems Accountable." In: *NDSS*. Vol. 9. 2009, pp. 341–353.

[13] Mohammad-Mahdi Bazm et al. "Side Channels in theCloud: Isolation Challenges, Attacks, and Countermeasures". In: (2017).

[14] BitsAbout.me. *BitsaboutMe is a new service that empowers you to reclaim control over your personal data, in order to better protect your privacy and to get a fair deal when sharing your personal data profile with trustworthy companies and institutions*. 2012. URL: `https://bitsabout.me`.

[15] Dan Boneh, Amit Sahai, and Brent Waters. "Functional Encryption: Definitions and Challenges". In: *TCC*. Vol. 6597. Lecture Notes in Computer Science. Springer, 2011, pp. 253–273.

[16] Stefan Brenner et al. "SecureKeeper: Confidential ZooKeeper using Intel SGX". In: *Middleware*. ACM, 2016, p. 14.

[17] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *OSDI*. USENIX Association, 1999, pp. 173–186.

[18] Miguel Castro et al. "Secure routing for structured peer-to-peer overlay networks". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 299–314.

[19] Yatin Chawathe et al. "Making gnutella-like P2P systems scalable". In: *SIGCOMM*. ACM, 2003, pp. 407–418.

[20] Jérémy Chotard et al. "Decentralized Multi-Client Functional Encryption for Inner Product". In: *ASIACRYPT (2)*. Vol. 11273. Lecture Notes in Computer Science. Springer, 2018, pp. 703–732.

[21] Ian Clarke et al. "Protecting Free Expression Online with Freenet". In: *IEEE Internet Computing* 6.1 (2002), pp. 40–49.

[22] Allen Clement et al. "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults". In: *NSDI*. USENIX Association, 2009, pp. 153–168.

[23] Chris Clifton et al. "Tools for Privacy Preserving Distributed Data Mining". In: *ACM SIGKDD Explorations Newsletter* 4 (Dec. 2002).

[24] Cozy Cloud. *A smart personal cloud to gather all your data*. 2012. URL: `https://cozy.io/en`.

[25] Fergus Dall et al. "CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.2 (2018), pp. 171–191.

[26] George Danezis et al. "Sybil-Resistant DHT Routing". In: *ESORICS*. Vol. 3679. Lecture Notes in Computer Science. Springer, 2005, pp. 305–318.

[27] Yves-Alexandre De Montjoye et al. "openpds: Protecting the privacy of metadata through safeanswers". In: *PloS one* 9.7 (2014), e98790.

[28] Digi.me. *See what your data can do for you with digi.me Private Sharing*. 2009. URL: `https://digi.me`.

[29] J.R. Douceur. "The Sybil attack". In: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*. 2002, pp. 252–260.

[30] Cynthia Dwork. "Differential Privacy". In: *ICALP (2)*. Vol. 4052. Lecture Notes in Computer Science. Springer, 2006, pp. 1–12.

[31] Ethereum. *Ethereum is the world's leading programmable blockchain*. URL: `https://ethereum.org`.

[32] Ethereum. *Ethereum White Paper*. URL: `https://github.com/ethereum/wiki/wiki/White-Paper`.

[33] Ben Fisch et al. "IRON: Functional Encryption using Intel SGX". In: *ACM Conference on Computer and Communications Security*. ACM, 2017, pp. 765–782.

[34] Craig Gentry. "A Fuly Homomorphic Encryption Scheme". PhD thesis. Standford University, 2009.

[35] Shafi Goldwasser et al. "Multi-input Functional Encryption". In: *EUROCRYPT*. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 578–602.

[36] Javier González et al. "A practical hardware-assisted approach to customize trusted boot for mobile devices". In: *International Conference on Information Security*. Springer. 2014, pp. 542–554.

[37] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. "A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems". In: *Computer Communication Review* 32.1 (2002), p. 82.

[38] Hamed Haddadi et al. "Personal Data: Thinking Inside the Box". In: *CoRR* abs/1501.04737 (2015).

[39] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30.

[40] Deokjin Kim et al. "SGX-LEGO: Fine-grained SGX controlled-channel attack and its countermeasure". In: *Computers & Security* 82 (2019), pp. 118–139.

[41] Seong Min Kim et al. "SGX-Tor: A Secure and Practical Tor Anonymity Network With SGX Enclaves". In: *IEEE/ACM Trans. Netw.* 26.5 (2018), pp. 2174–2187.

[42] Ramakrishna Kotla et al. "Zyzzyva: Speculative Byzantine fault tolerance". In: *ACM Trans. Comput. Syst.* 27.4 (2009), 7:1–7:39.

[43] Riad Ladjel et al. "A manifest-based framework for organizing the management of personal data at the edge of the network". In: *ISD*. 2019.

[44] Riad Ladjel et al. "Trustworthy Distributed Computations on Personal Data Using Trusted Execution Environments". In: *TrustCom*. 2019.

[45] Saliha Lallali et al. "Supporting secure keyword search in the personal cloud". In: *Information Systems* 72 (2017), pp. 1–26.

[46] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401.

[47] Chris Lesniewski-Laas and M. Frans Kaashoek. "Whanau: A Sybil-proof Distributed Hash Table". In: *NSDI*. USENIX Association, 2010, pp. 111–126.

[48] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity". In: *ICDE*. IEEE Computer Society, 2007, pp. 106–115.

[49] Xiaolei Li et al. "DroidVault: A Trusted Data Vault for Android Devices". In: *ICECCS*. IEEE Computer Society, 2014, pp. 29–38.

[50] Chu Yee Liau et al. "Efficient Distributed Reputation Scheme for Peer-to-Peer Systems". In: *Human.Society@Internet 2003*. Vol. 2713. Lecture Notes in Computer Science. Springer, 2003, pp. 54–63.

[51] Linaro. *OP-TEE*. 2014. URL: `https://wiki.linaro.org/WorkingGroups/Security/OP-TEE`.

[52] Yehuda Lindell and Ben Riva. "Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries". In: *ACM Conference on Computer and Communications Security*. ACM, 2015, pp. 579–590.

[53] Litecoin. *Litecoin — The cryptocurrency for payments*. URL: `https://litecoin.org/`.

[54] Julien Loudet, Iulian Sandu Popa, and Luc Bouganim. "DISPERS: Securing Highly Distributed Queries on Personal Data Management Systems". In: *PVLDB* 12.12 (2019), pp. 1886–1889.

[55] Julien Loudet, Iulian Sandu Popa, and Luc Bouganim. "SEP2P: Secure and Efficient P2P Personal Data Processing". In: *EDBT*. OpenProceedings.org, 2019, pp. 145–156.

[56] Ashwin Machanavajjhala et al. "*L*-diversity: Privacy beyond *k*-anonymity". In: *TKDD* 1.1 (2007), p. 3.

[57] Petar Maymounkov and David Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric". In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 53–65.

[58] Brian McGillion et al. "Open-TEE - An Open Virtual Trusted Execution Environment". In: *TrustCom/BigDataSE/ISPA (1)*. IEEE, 2015, pp. 400–407.

[59] Meeco. *Meeco — the distributed technology powering consent and personal data*. 2012. URL: `https://www.meeco.me`.

[60] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[61] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *CRYPTO*. Vol. 293. Lecture Notes in Computer Science. Springer, 1987, pp. 369–378.

[62] MesInfos. *The MesInfos project explores and implements the self data concept in France*. 2012. URL: `http://mesinfos.fing.org/english`.

[63] Prateek Mittal, Matthew Caesar, and Nikita Borisov. "X-Vine: Secure and Pseudonymous Routing in DHTs Using Social Networks". In: *NDSS*. The Internet Society, 2012.

[64] Sonia Ben Mokhtar et al. "X-search: revisiting private web search using intel SGX". In: *Middleware*. ACM, 2017, pp. 198–208.

[65] Matei Ciobanu Morogan and Sead Muftic. "Certificate Management in Ad Hoc Networks". In: *SAINT Workshops*. IEEE Computer Society, 2003, pp. 337–341.

[66] Jose L. Muñoz et al. "Certificate revocation system implementation based on the Merkle hash tree". In: *Int. J. Inf. Sec.* 2.2 (2004), pp. 110–124.

[67] MyData.org. *MyData Global's mission is to empower individuals by improving their reight to self-determination regarding their personal data*. 2014. URL: `https://mydata.org`.

[68] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: `https://www.bitcoin.com/bitcoin.pdf`.

[69] Nextcloud. *The self-hosted productivity platform that keeps you in control*. 2016. URL: `https://nextcloud.com`.

[70] Inc. Novathings. *helixee — The French cloud that respects your privacy.* URL: `http://www.helixee.me/home/`.

[71] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. "Reaching Agreement in the Presence of Faults". In: *J. ACM* 27.2 (1980), pp. 228–234.

[72] Riccardo Pecori. "S-Kademlia: A trust and reputation method to mitigate a Sybil attack in Kademlia". In: *Computer Networks* 94 (2016), pp. 205–218.

[73] Sandro Pinto and Nuno Santos. "Demystifying Arm TrustZone: A Comprehensive Survey". In: *ACM Comput. Surv.* 51.6 (2019), 130:1–130:36.

[74] Christian Priebe, Kapil Vaswani, and Manuel Costa. "EnclaveDB: A Secure Database Using SGX". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 264–278.

[75] Sylvia Ratnasamy et al. *A scalable content-addressable network*. Vol. 31. 4. ACM, 2001.

[76] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. "Anonymous connections and onion routing". In: *IEEE Journal on Selected Areas in Communications* 16.4 (1998), pp. 482–494.

[77] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC* 8446 (2018), pp. 1–160.

[78] Antony Rowstron and Peter Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems". In: *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer. 2001, pp. 329–350.

[79] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. "The Earth Mover's Distance as a Metric for Image Retrieval". In: *International Journal of Computer Vision* 40.2 (2000), pp. 99–121.

[80] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not". In: *TrustCom/BigDataSE/ISPA (1)*. IEEE, 2015, pp. 57–64.

[81] Eyad Saleh et al. "Processing over encrypted data: between theory and practice". In: *ACM SIGMOD Record* 45.3 (2016), pp. 5–16.

[82] Pierangela Samarati. "Protecting Respondents' Identities in Microdata Release". In: *IEEE Trans. Knowl. Data Eng.* 13.6 (2001), pp. 1010–1027.

[83] Samsung. *Whitepaper: Samsung Knox Security Solution*. May 2017. URL: `https://images.samsung.com/is/content/samsung/p5/global/business/mobile/SamsungKnoxSecuritySol` `pdf`.

[84] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[85] Rashid Sheikh, Durgesh Kumar Mishra, and Beerendra Kumar. "Secure Multiparty Computation: From Millionaires Problem to Anonymizer". In: *Information Security Journal: A Global Perspective* 20.1 (2011), pp. 25–33.

[86] Atul Singh et al. "Eclipse Attacks on Overlay Networks: Threats and Defenses". In: *INFOCOM*. IEEE, 2006.

[87] Anurag Singla and Christopher Rohrs. *Ultrapeers: Another Step Towards Gnutella Scalability*. 2001.

[88] SpiderOak. *SpiderOak Share provides a secure way to exchange and sync your files using No Knowledge Encryption.* 2018. URL: `https://spideroak.com/spideroak-share`.

[89] Ion Stoica et al. "Chord: A scalable peer-to-peer lookup service for internet applications". In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[90] He Sun et al. "TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices". In: *DSN*. IEEE Computer Society, 2015, pp. 367–378.

[91] Latanya Sweeney. "k-Anonymity: A Model for Protecting Privacy". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pp. 557–570.

[92] Sync. *Sync's end-to-end encrypted storage platform and apps ensure that only you can access your data in the cloud.* 2011. URL: `https://www.sync.com`.

[93] ARM Security Technology. *Building a Secure System using TrustZone Technology.* Tech. rep. ARM, Dec. 2008.

[94] Dai Hai Ton That et al. "PAMPAS: Privacy-Aware Mobile Participatory Sensing Using Secure Probes". In: *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*. ACM. 2016, p. 4.

[95] *The Gnutella Protocol Specification v0.4*.

[96] Inc. The Tor Project. *Tor: Overview.* URL: `https://2019.www.torproject.org/about/overview.html.en`.

[97] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. "Privacy-Preserving Query Execution using a Decentralized Architecture and Tamper Resistant Hardware". In: *EDBT*. OpenProceedings.org, 2014, pp. 487–498.

[98] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. "Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture". In: *ACM Trans. Database Syst.* 41.3 (2016), 16:1–16:43.

[99] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. "A survey of DHT security techniques". In: *ACM Computing Surveys (CSUR)* 43.2 (2011), p. 8.

[100] Jo Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution". In: *Proceedings of the 27th USENIX Security Symposium*. See also technical report Foreshadow-NG [**weisse2018foreshadowNG**]. USENIX Association, Aug. 2018.

[101] Peng Wang, Ivan Osipkov, and Yongdae Kim. *Myrmic: Secure and robust DHT routing.* Tech. rep. 2007.

[102] Qiyan Wang and Nikita Borisov. "Octopus: A Secure and Anonymous DHT Lookup". In: *ICDCS*. IEEE Computer Society, 2012, pp. 325–334.

[103] Zcash. *Zcash is a privacy-protecting, digital currency built on strong science.* URL: `https://z.cash/`.

[104] Zcash. *Zcash White Paper.* URL: `https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf`.

[105] Ben Y. Zhao et al. "Tapestry: a resilient global-scale overlay for service deployment". In: *IEEE Journal on Selected Areas in Communications* 22.1 (2004), pp. 41–53.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Requêtes Distribuées Respectueuses de la Vie Privée sur Nuages Personnels

Mots clés : Protection de la Vie Privée, Nuage Personnel, Pair-à-Pair, Système Distribué

Résumé : Dans un contexte où nous produisons de plus en plus de données personnelles et où nous contrôlons de moins en moins comment et par qui elles sont utilisées, une nouvelle manière de les gérer voit le jour: le « nuage personnel ». En partenariat avec la jeune pousse française Cozy Cloud (https://cozy.io) qui développe une telle technologie, nous définissons à travers ces travaux une manière collaborative d'interroger ces nuages personnels qui respecte la vie privée des utilisateurs.
Pour y parvenir nous détaillons dans cette thèse trois contributions : (1) un ensemble de quatre prérequis que tout protocole doit respecter dans ce contexte particulier : l'*aléa imposé* qui empêche un attaquant d'influencer le déroulement de l'exécution, la *dispersion des connaissances* qui assure qu'aucun participant ne possède trop d'informations, l'*atomicité des tâches* qui diminue au maximum le rôle joué par chaque participant directement impliqué dans l'exécution et les *communications cachées* pour protéger l'identité des participants et les informations échangées ; (2) `SEP2P` un protocole se basant sur une table de hashage distribuée et `CSAR`, un protocole permettant de générer un nombre aléatoire, afin de générer une liste aléatoire et vérifiable d'acteurs en accord avec le premier prérequis ; et (3) `DISPERS` un protocole qui applique les trois derniers prérequis et découpe l'exécution d'une requête de sorte à minimiser les informations accédées par chaque acteur pour minimiser l'impact d'une fuite au cas où un attaquant aurait été sélectionné parmi ces mêmes acteurs.

Title : Distributed and Privacy-Preserving Queries on Personal Clouds

Keywords : Privacy-preserving, Personal Cloud, Peer-to-Peer, Distributed System

Abstract : In a context where we produce more and more personal data and where we control less and less how and by whom they are used, a new way of managing them is on the rise: the "personal cloud". In partnership with the french start-up Cozy Cloud (https://cozy.io) that is developing such technology, we propose through this work a way of collaboratively querying the personal clouds while preserving the privacy of the users.
We detail in this thesis three contributions to achieve this objective: (1) a set of four requirements any protocol has to respect in this particular context: *imposed randomness* to prevent an attacker from influencing the execution of a query, *knowledge dispersion* to prevent any node from concentrating information, *task atomicity* to split the execution in as many independent tasks as necessary and *hidden communications* to protect the identity of the participants as well as the content of their communications; (2) `SEP2P` a protocol leveraging a distributed hash table and `CSAR`, another protocol that generates a verifiable random number, in order to generate a random and verifiable list of actors in accordance with the first requirement; and (3) `DISPERS` a protocol that applies the last three requirements and splits the execution of a query so as to minimize the impact of a leakage (in case an attacker was selected as actor) by providing to each actor the minimum amount of information it needs in order to execute its task.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France