Highly distributed and privacy-preserving queries on personal data management systems

Luc Bouganim, Julien Loudet, Iulian Sandu Popa

VLDB manuscript No.
(will be inserted by the editor)

Highly Distributed and Privacy-Preserving Queries on Personal Data Management Systems

Luc Bouganim · Julien Loudet · Iulian Sandu Popa

Received: date / Accepted: date

Abstract Personal Data Management System (PDMS) solutions are flourishing, boosted by smart disclosure initiatives and new regulations. PDMSs allow users to easily store and manage data directly generated by their devices or resulting from their (digital) interactions. Users can then leverage the power of their PDMS to benefit from their personal data, for their own good and in the interest of the community. The PDMS paradigm thus brings exciting perspectives by unlocking novel usages, but also raises security issues. An effective approach, considered in several recent works, is to let the user data distributed on personal platforms, secured locally using hardware and/or software security mechanisms. This paper goes beyond the local security issues and addresses the important question of securely querying this massively distributed personal data. To this end, we propose DISPERS, a fully-distributed PDMS peer-to-peer architecture. DISPERS allows users to securely and efficiently share and query their personal data, even in the presence of malicious nodes. We consider three increasingly powerful threat models and derive, for each, a security requirement that must be fulfilled to reach a lower-bound in terms of sensitive data leakage: (i) *hidden communications*, (ii) *random dispersion of data*, and (iii) *collaborative proofs*. These requirements are incremental and respectively resist spied, leaking or corrupted nodes. We show that the expected security level can be guaranteed with near certainty and validate experimentally the efficiency of the proposed protocols, allowing for adjustable trade-off between the security level and its cost.

Keywords Distributed systems, Privacy, Personal Data Management System, Peer-to-peer query processing.

L. Bouganim · J. Loudet · I. Sandu Popa
INRIA Saclay, 1 rue H. d'Estienne d'Orves, 91120 Palaiseau, France,
U. of Versailles, 45 avenue des États-Unis, 78035 Versailles, France,
E-mail: <fname.lname>@inria.fr, <fname.lname>@uvsq.fr

1 Introduction

Personal Data Management Systems (PDMSs): The time for individualized management and control over one's personal data has arrived. Thanks to smart disclosure initiatives (see MyData Global [46]) and new regulations (e.g., the General Data Protection Regulation [24]), users can access their personal data from the companies or government agencies that collected them. Concurrently, Personal Data Management System (PDMS) solutions are flourishing [4] both in the academic area (e.g., Personal Information Management Systems, Personal Data Servers [1], Personal Data Stores [20, 47], Personal Clouds [35]) and industry [19,48,65]. Their goal is to offer users a platform – that acts as a single point of entry – where they can easily store and manage the data generated by their devices (quantified-self data, smart home data, photos, etc.) and resulting from their interactions (social interaction data, health, banking, telecom, etc.). Users can then leverage the power of their PDMS to benefit from their personal data for their own good and in the interest of the community. Thus, the PDMS paradigm promises to unlock innovative uses: PDMS users can contribute their personal data and benefit from this globally contributed data through distributed queries. These queries can compute recommendations [69], enable participatory studies [53][47], deliver relevant information to users based on their profile [70], or consider ad-hoc cohorts for scientific purposes, in the spirit of the recent concept of *data altruism* [23].

Trustworthy PDMSs under owner's control: These exciting perspectives should not eclipse the security issues raised by this paradigm. Indeed, each PDMS potentially stores the entire digital life of its owner, thereby increasing the impact of a leakage. It is therefore risky to centralize all user data in powerful servers as these servers become highly desirable targets for attackers: huge amounts of personal data belonging to millions of individuals could be leaked or lost ([78]

records more than 10^{10} email addresses in data breaches). Besides, centralized solutions make little sense in the PDMS context in which data are naturally distributed at the users' side [32]. Alternatively, recent works [4,6,20,34,35,47,65] propose to let the user data distributed on personal trustworthy platforms under the users' control. Such platforms can be built thanks to (i) *a secure hardware component* providing a *Trusted Execution Environment*, such as smart cards [1], secure micro-controllers [4,5,35], ARM TrustZone [52], or Intel SGX [55] and/or (ii) *specific software* (e.g., minimal Trusted Computing Base and information flow control [37, 55,13]). In this paper, we follow this approach and consider that a PDMS is a dedicated device possessed by the user and secured with some security mechanisms.

A fully decentralized approach: As in many academic and commercial solutions [65], we assume that PDMSs offer a rather good connectivity and availability like home-cloud solutions [4,19,33,48,65] built on plug computers connected to the Internet. Thus, PDMSs can establish peer-to-peer (P2P) connections with other PDMSs and participate in a distributed computation by providing part of the data, thus acting as *data sources* and/or performing part of the processing, thus acting as *data processors*. Hence, we focus on architectures based on a full distribution of PDMSs (indifferently called nodes) acting as data sources and/or data processors through P2P interactions. Solutions requiring a re-centralization of distributed personal data during processing must be discarded as this dynamically creates a concentration of personal data and leads to a risk similar to that of centralized servers.

Motivating example and naive execution: Let us consider the following query: "average number of sick leave days in 2022 for French teachers" necessary to a study on illness at work. This query requires effective targeting of the most appropriate data sources (i.e., French teachers) to ensure both the result relevance and the system scalability. We thus propose a targeting based on *user profiles* (e.g., user's profession or country) provided by each PDMS. These profiles are stored in a distributed index leveraging classical Distributed Hash Tables (DHT) which offer efficient, scalable, and fault-tolerant decentralized storage and node location facilities. A naive execution strategy can then be easily conceived (see Figure 1): the querier (1) retrieves the list of PDMS users who are French and teacher thanks to the distributed profile index; (2) sends them a local query (number of sick leave days in 2022); and (3) aggregates their local results.

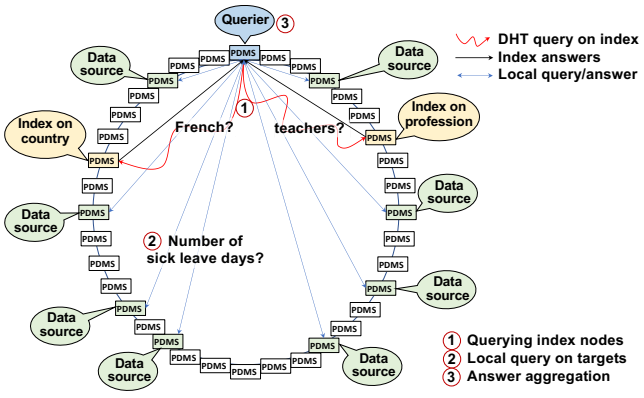

Fig. 1: Motivating example and naive strategy

Goal and challenges: Our goal is to propose a large-scale, fully decentralized PDMS system that (i) efficiently targets the nodes pertinent for a query based on user profiles and (ii) aggregates local results to compute non-sensitive global results (e.g, statistics), useful for the envisioned applications, while (iii) protecting user privacy, i.e., protecting the participants profiles, the local results, and most importantly, their associations. The naive strategy presented above cannot achieve this goal given the central role of the querier who has access to the sensitive data of the query and, moreover, is a potential bottleneck. Indeed, although we consider trustworthy PDMSs, we must admit that no security measure can be considered unbreakable: some nodes (including the querier) can be spied on, leaking data or fully corrupted. Even worse, these nodes can collude and may very well be indistinguishable from honest nodes, acting as covert adversaries [7]. The challenge is then to define execution protocols that solely rely on PDMS nodes and offer strong guarantees in terms of data leakage. Moreover, our aim in this paper is to understand what guarantees can be achieved when PDMS nodes work on clear-text data, allowing generic computations, accurate (noise-free) results, and greatly simplifying node failure management (see Section 8 and [45]).

Related works: Both P2P systems and secure distributed computations have been hot research topics for many years. However, we are not aware of any decentralized solution allowing to securely target nodes based on user profiles — a critical feature for the considered applications. Regarding the data aggregation itself, existing works related to Multi-Party Computation protocols (MPC) or Differential Privacy (DP) cannot be applied in our context. On the one hand, MPC typically leverages centralized or federated architectures in which a handful of powerful servers hold large collections of user data (e.g., SMCQL [9], Conclave [73] and Obscure [27]) or collect user data at query time (e.g., Prio [18]). These solutions may offer strong security guarantees (both data confidentiality and result correctness) but are not adapted to decentralized execution with thousands of participants or lack generality w.r.t. the computed function. On the other hand, DP requires a central trusted third party to obfuscate sensitive information. Local DP (LDP) does not need such a third party since obfuscation is performed at the level of individual sources but it does require a huge number of participants to reduce the impact of added noise [3]. The queries considered here involve a few thousand participants to be statistically significant, i.e., too many for MPC to be effec-

tive and too few to reduce the impact of noise with LDP. Related work is discussed in more detail in Section 8.

Problem formulation: Given the potential presence of covert, corrupted and colluding nodes, and the inability to use MPC or LDP, it becomes impossible to provide efficient, leak-free targeted query execution involving only PDMSs that work on clear text data. Indeed, communications can be spied, nodes storing the distributed index and data processors may be corrupted, thus potentially leaking indexes or local results from data sources. The problem is then to define the lower-bound on leakage in this context and to propose an architecture and associated protocols that are *secure*, i.e., that reach this lower-bound, yet *efficient*, i.e., that keep the security overheads low.

High level idea: In this paper, we propose DISPERS (DIStributed Privacy-presERving querieS) which can: (i) be applied to very large P2P systems; (ii) select specific data source nodes to achieve query pertinence; (iii) reach a lower-bound on leakage in the presence of colluding nodes with *near-certainty*, i.e., with a very high and adjustable probability; (iv) adjust the trade-off between security level and security cost. Thus, DISPERS enables generic, efficient and scalable P2P data computations while still providing users with strong confidentiality guarantees. We design DISPERS in an incremental way by considering increasingly stronger threat models. For each of them, we derive a *security requirement* that must be satisfied to reach the lower-bound on leakage (see Figure 2). More precisely, we consider:

(i) *spied nodes* when the attacker can spy on PDMS communications, leading to the ***H**idden communication* requirement and the DISPERS^{H} protocol in which communications are protected through encryption and anonymization;

(ii) *leaking nodes* when the attacker can, additionally, observe the internal states of the PDMS, leading to the ***R**andom dispersion of data* requirement and the DISPERS^{HR} protocol in which data-at-rest, i.e. the indexes, are protected using secret sharing [63] and data-in-use are protected through task compartmentalization; and finally,

(iii) *corrupted nodes* when the attacker fully controls some PDMSs (thus can additionally alter its behavior), leading to the ***C**ollaborative proofs* requirement and to the DISPERS^{HRC} protocol, in which a contributing or indexing node answers a request for sensitive data only if it obtains a collaborative proof that the request is justified.

Contributions: We make the following contributions:

(1) We propose a P2P architecture of PDMSs relying on classical DHTs, a data model allowing to efficiently target relevant nodes, and a query model to express the queries of interest. We show, using a functional but insecure naive protocol, that the architecture enables the considered applications in a fully distributed fashion.

(2) We analyze three possible threat models, define and justify the lower-bound on leakage.

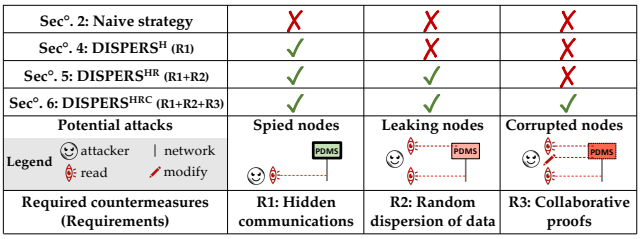

Fig. 2: Paper roadmap

(3) We derive from each threat model a security requirement and propose suitable execution protocols that satisfy each and reach the lower-bound on leakage: DISPERS^{H}, DISPERS^{HR} and DISPERS^{HRC} to, respectively, resist spied, leaking or corrupted nodes while considering efficiency as a second major goal.

(4) We provide a security analysis for each threat model that shows that the lower-bound on leakage can be reached with near-certainty, even in the worst case scenario of large collusion attacks from fully corrupted nodes.

(5) We show the feasibility of our approach by experimentally evaluating the effectiveness of our protocols. We show that they have a reasonable, tunable security overhead and are fully scalable with the number of colluding nodes.

Achievable guarantees: In the worst-case scenario where an attacker masters a large number of covert corrupted nodes, our proposal (i) fully protects the sensitive distributed profile index required to target nodes; (ii) ensures that the query processing cannot leak more than a fraction of the IP addresses or local results of the targeted nodes, proportional to the percentage of colluding nodes that the attacker controls in the system (which is the best security that can be provided in our context); (iii) precludes the association of local results with the nodes that issued them.

Outline: The paper[1] is organized as follows: Sections 2 and 3 respectively present the architecture and threat models, allowing to state the problem at hand. The next three sections progressively propose protocols for each threat model, along with the associated security analysis. Their performance is evaluated experimentally in Section 7. Section 8 and 9 discusses related work and DISPERS limitations. We conclude in Section 10. Appendices A and B provide useful background on cryptography and distributed systems.

[1] This paper is based on previous studies [38,39,40]: in [40], we showed that the execution of a P2P query can indeed rely exclusively on data processor nodes if and only if they are selected in a verifiable random way, which cannot be influenced by corrupted nodes. [39] is a demonstration of DISPERS architecture and applications. [38] is a PhD manuscript. It includes implementation details of the protocols proposed in this paper and describes a proof-of-concept implementation of the most advanced protocol into the Cozy Cloud product [19].

2 Architectural Design & Naive Protocol

We detail the architecture of DISPERS and show that our approach is doable with a naive protocol despite its limitations.

2.1 Fully-Distributed System

DISPERS is a fully-distributed P2P system relying solely on PDMSs to enable the envisioned applications. Therefore, each node is potentially a *Data Source* that provides its data, and/or a *Data Processor* providing part of the required processing by fulfilling a *role*. The first obvious role is that of the *Querier* (Q) initiating the distributed processing. Table 1 summarizes the roles defined throughout the paper while table 2 lists the associated notations and abbreviations (the last column specifies the section where the role/abbreviation first appears).

Relying on a P2P system poses several challenges, such as integrating new nodes, maintaining a consistent global state, making nodes interact, handling churn or maintaining some metadata. It thus requires a communication overlay allowing for efficient node discovery, data indexing and search. These requirements naturally lead to DHTs (see Appendix B) which DISPERS leverages as a basis for efficient and scalable communications. Currently, we implemented the Chord DHT and used it for the experiments in Section 7.

Role	Description of the role (first defined in §)	§
Q	the **Querier** initiates the distributed processing.	2.1
T	**Targets** are nodes whose profile matches *tp*.	2.3
CI	a **Concept Indexer** responsible for a concept c_i stores the list of *TIPs* of the nodes possessing c_i in their profile.	2.5
NP	a **Node Proxy** inserts a concept in lieu of a node.	4.2
P	a **Proxy** forwards a communication to the next P / NP.	4.2
BP	a **Before Proxy** forwards the communications going to a Target in order to hide its *TIP* from an attacker.	4.3
AP	an **After Proxy** forwards the communications coming from a Target in order to hide its *TIP* from an attacker.	4.3
W	**Workers** supplant Q to transfer *lq* to Ts, to aggregate their results, and to send back the aggregate to Q.	4.3
AS	the **Actor Selector** is a node chosen randomly, in charge of selecting the query actors.	4.3
PS	**Profile Samplers** reconstruct $\overline{TIP}$, apply $\overline{tp}$, sample the resulting Targets, and sends associated data to TFs.	5.3
TF	**Target Finders** reconstruct *TIP* only for the sampled Targets and send them anonymously the encrypted *lq*.	5.3
DA	**Data Aggregator** aggregate the local results using $\overline{aq}$.	5.3
FDA	the **Final DA** performs the final aggregation, sent to Q.	5.3
CIL	**CI's Legitimate nodes** verify and attest that some neighbor node is actually the CI indexing a concept.	6.4
QL	**Q's Legitimate nodes** verify and attest the query validity (query budget), and bootstrap the random selection of actors by generating a verifiable random number.	6.4
ASL	**AS's Legitimate nodes** generate the list of actors, sign it and, through their signatures, link it with the query.	6.4

Table 1: Roles definitions

2.2 Data Model

Given the fully-distributed nature of our system, each query can potentially involve a large number of nodes. To ensure scalability and resilience, it is crucial to limit the number of nodes involved in a computation. However, poorly choosing the participants can lead to uninteresting or no results at all. Thus, each PDMS publishes a *profile* that is used to limit the participants to those that are relevant.

A *profile* p is a set of concepts: $p = \{c_1, \ldots, c_n\}$. Each *concept* c_i is the concatenation of metadata terms m_i describing its semantics, and a single value v: $c_i = m_1|m_2|\ldots|m_p|v$. Examples are: `location`$|$*Lyon*, `sex`$|$*male*. Multiple metadata terms can be used to indicate a concept at different granularities, allowing for a structured organization. For instance, `location`$|$`city`$|$*Lyon*.

The profile is an accurate description of the PDMS owner. It can be generated automatically by the PDMS according to the data it contains, and/or manually by the owner by selecting attributes she finds fitting. Besides the profile, the user may contribute with part of her stored personal data (e.g., rated movies, physical activities statistics, etc.) for aggregated queries from other users. Note that the proposed protocols are not dependent on the precise semantics of concepts and profiles. We only assume that the user's profile and data are structured and thus can be queried. Obviously, both are sensitive and as such require protection.

2.3 Query Model

We consider three iconic applications based on large user communities that benefit greatly from the PDMS paradigm: (i) *distributed query processing on personal data of large sets of individuals* [69,47], in which users contribute with their personal data and query the globally contributed data (e.g., computing recommendations, participatory studies); (ii) *profile or subscription-based data diffusion apps* [70], where users provide profiles to selectively receive relevant information; (iii) *mobile participatory sensing apps* [53], in

Abb.	Description of the abbreviation (first defined in §)	§
c_i	a **concept** c_i is a set of metadata terms m_i describing its semantics, and a value v: $c_i = m_1\|m_2\|\ldots\|m_p\|v$.	2.2
p	a **profile** is a set of concepts: $p = \{c_1, \ldots, c_n\}$.	2.2
tp	the **Target Profile** is a logical expression of concepts indicating which nodes qualify to answer a query.	2.3
lq	the **Local Query** is computed locally by each Target.	2.3
aq	the **Aggregate Query** is applied over the *lq* results.	2.3
TIP	**Target IP**, i.e., IP address of a Target node.	2.5
$\overline{x}$	denotes the **pseudonym** associated with a data x.	5.1
sel	the **selector** is used with multiple PSs and TFs.	5.4
CTID	**Concept-Target Identifier** $CTID = \texttt{hash}(k_{pub}\,|\,c_i\,|\,RND)$.	5.4

Table 2: Abbreviations & notations

which mobile users produce sensed geo-localized data using their smartphones or vehicular systems (e.g., traffic, noise) to compute spatially aggregated statistics for the whole community. We only detail queries falling into the first class since it is almost the combination of the other two: finding the relevant subset of nodes as with class (ii) and computing the query result based on the data supplied as with class (iii).

We use three examples to illustrate a query definition:
(i) a **closed list of items query** considers a list of items defined in the query and delivers statistics computed on that list: e.g., "find the average rating of these movies: {Dune, Star Wars, Drive, Inception, Skyfull} as given by researchers or professors living in Paris";
(ii) an **open item list query** with the goal to determine a list of items with specific characteristics (best, worst, etc.) verified by the participants: e.g., "get the top-10 ranked movies as chosen by researchers or professors living in Paris";
(iii) a **statistical query**: e.g., "average number of sick leave days in 2020 of researchers or professors living in Paris".

All these queries expose a *Target Profile* (*tp*), which is a logical expression of concepts indicating which nodes, called *Targets* should be involved in the computation (matching *tp*). For the examples above, $tp = (\texttt{location}|\texttt{city}|\textit{Paris}) \wedge ((\texttt{profession}|\textit{researcher}) \vee (\texttt{profession}|\textit{professor}))$.

The *Local Query (lq)* specifies the data that each Target must provide for the computation: e.g., "ratings of a set of movies" or "number of sick leave days in 2020".

Finally, the *Aggregate Query (aq)*, is a classical aggregate expression (average, top-10 in our examples, or count, min, max, group-by, etc.) applied over the *lq* results.

We thus define a query q as a triplet: $q = (tp, lq, aq)$.

More complex queries can be considered if their execution can be decomposed in a *Local* and an *Aggregate Query*. Query expressiveness issues are left for future work, as we focus on privacy preservation and efficient query evaluation.

2.4 Number of Targets, Sampling and Query Budget

The number of Targets for a given Target Profile ranges from low (very specific *tp*) to very high (popular or vague *tp*). Query results computed with too few Targets may become sensitive and thus, should not be computed: a threshold value should be defined, which may depend on the application [62].

For queries involving too many Targets, DISPERS operates a *sampling phase* to randomly select a subset of them with the immediate benefit of alleviating the system load during query processing. Note that this does not necessarily degrade the quality of the final result: mathematical statistics — Hoeffding's inequality [30] for instance — show that an adequate sample is as representative as the entire population. In addition, sampling increases the system security by reducing the risk of exposure. Note that the query result includes the number of Targets involved in the computation.

However, to negate the security benefits of this sampling phase, an attacker could execute the same query over and over again. To reduce the attractiveness of this attack, and, more generally, to reduce the risks of data disclosure, we introduce a *query budget*, a system parameter that applies to all nodes and sets a limit to the number of queries and/or Targets allowed per time period. Once exhausted, the node cannot issue any more queries.

2.5 Distributed Concept Index

To efficiently determine the list of matching Targets, each node profile must be indexed. We can leverage the DHT (see Appendix B) for efficient storage and retrieval of these indexes: the *concept index* is the association between a concept c_i, and the list of node addresses, called *TIP* (for Target IP), whose owner's profile includes c_i.

Thus, for each concept c_i in its profile, a node performs a $\texttt{store}(c_i, \textit{TIP})$ DHT operation that adds *TIP* to the concept index of c_i. Since the DHT uniformly distributes the indexed data among the nodes, each node may be responsible for indexing one or several concepts. Updating a node profile is done in a similar way by leveraging the DHT.

To find the Targets, the Querier performs a $\texttt{lookup}(c_i)$ DHT operation for each concept c_i in *tp*. The nodes responsible for storing each concept are called *Concept Indexer* (CI). The CIs store the list of *TIPs* of the nodes possessing c_i in their profile and send that list whenever asked by Q.

2.6 Naive Protocol

We describe below a first naive protocol (see Fig. 3) demonstrating that the proposed query model can be supported:

Protocol 1. Naive protocol

1. Q looks up in the DHT the CIs indexing the different concepts composing the Target Profile (*tp*).
2. The CIs send to Q the *TIPs* lists for each queried concept.
3. Q applies *tp* on the lists to find the Targets (Ts), samples the Ts and sends to each sampled T the Local Query (*lq*).
4. The Ts apply *lq* and send back their local results.
5. Q finally applies the Aggregate Query (*aq*) on the local results to obtain the final result.

Obviously, this naive protocol has major shortcomings: (i) the Querier centralizes the entire execution flow and becomes a bottleneck; and (ii) no form of protection is offered (either for data, metadata, or budget controls). The first shortcoming relates to efficiency which can be a detrimental factor to system adoption and is thus addressed in the forthcoming protocols. The second is related to security and closely tied to the attacks faced by the system. We thus study next in detail the threat models considered in this work.

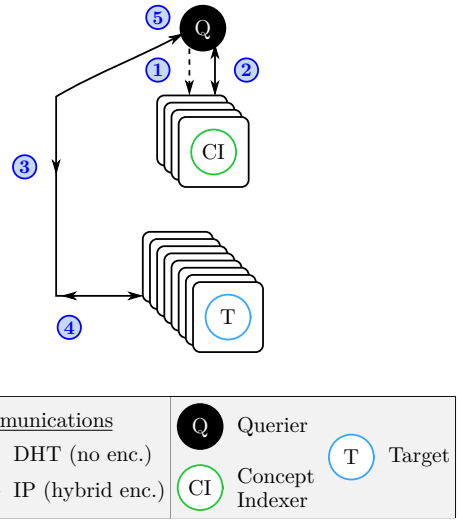

Fig. 3: Naive protocol

3 Threat Models, Leakages and Problem Formulation

This section presents our security assumptions, the threat models considered, defines the lower-bound on leakage and formulate the problem more precisely.

3.1 Threat Models

Because of the distributed nature of our system, the attacks can either target the communications (see Assumption 1) or the nodes themselves (see Assumptions 2 & 3):

Assumption 1 *An attacker can observe the content and the metadata of communications between a predefined subset of nodes in the network, i.e., the subset cannot change during a distributed computation.*

This assumption indicates that an attacker can spy on communications between some nodes but cannot spy on the entire network ("state-size attacks"). As a consequence, although the subset of observed nodes can change over time, this cannot be done in real-time (i.e., during a query processing). Note that the solutions we provide can be tuned to resist to a very large percentage of spied nodes (see Section 4.5).

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

Without this assumption, an attacker can emulate fake nodes and conduct a Sybil attack [22]: controlling a large portion of nodes (thus those processing the query) and thwarting any countermeasure. We can rely on a classical Public Key Infrastructure (PKI) to deliver these certificates. We do not require the PKI to be *online* since the certificate is attached to the PDMS device, not to the device owner.

Assumption 3 *Each PDMS is locally secured using some security mechanisms (hardware, software or others means).*

This third assumption is reasonable since a PDMS is supposed to store the entire digital life of its owner and thus, must be properly secured. There are several ways to provide local guarantees: (i) relying on Secure Hardware technology [4,6] — Intel SGX is now present in most recent processors and ARM TrustZone in most mobile devices; (ii) using software and/or hardware protections enforced by the operating system (e.g., IOS [26], Android [25], seL4 [29]); or even (iii) other means (e.g., context of use, trust, laws).

Common security measures include: safe-keeping of secrets, tamper-resistance, isolation and attestation of the executed code [60]. In particular, the *isolation* property means that the executed code and the data manipulated cannot be observed by an unauthorized process. This is particularly interesting in our context since it represents the building block to achieve data confidentiality.

However, no security mechanism is unbreakable: [49] surveys several attacks to leak part — if not all — of the computations performed within an SGX enclave. Resourceful attackers can conduct highly advanced *lab attacks* [52], to compromise a PDMS using an expensive equipment.

In order to account for the diversity of security mechanisms and their possible shortcomings, we focus on three threat models of increasing difficulty — i.e., considering that a PDMS offers less and less protection. Proceeding incrementally allows us to gradually expose the different techniques we employ to protect users and their data. This also means that the Sections 4, 5 and 6, devoted to each threat model, cannot be taken independently as we reuse these techniques. The threat models are as follows:

Threat Model 1 - Tamper-proof. Under this model, also called *fully honest*, we assume that the PDMS device cannot be tampered with: attackers cannot access any data stored or manipulated by their PDMS. They can, however, observe the communications as they occur outside of this secure environment (see Assumption 1). Nodes that are eavesdropped on are referred to as *spied nodes*. This threat model is somewhat optimistic (unless e.g., some strong legal constraints apply), but it allows both as a baseline and for the gradual introduction of security requirements and associated countermeasures.

Threat Model 2 - Passive attack. Under this model, also known as *honest-but-curious*, in addition to observing the communications, attackers can access, *without altering*, the data stored or manipulated by their PDMS — hence the "passive" attack (read only). Nodes that suffer a passive attack are referred to as *leaking nodes*.

Threat Model 3 - Active attack. Under this model, also known as *malicious*, an attacker has complete control of her

own PDMS, having bypassed the local security mechanisms and thus can access and alter both the data stored or manipulated and the code executed on her PDMS (but cannot falsify a certificate to contradict Assumption 2). Nodes suffering an active attack are called *corrupted nodes*.

Attacker model. We consider every owner of a PDMS as a potential attacker, especially the querier. Attackers behave as covert adversaries [7], i.e., they only derive from the protocol to obtain private information if they cannot be detected as otherwise they would be excluded. An attacker can be one or several colluding malicious users and thus, de facto, control more than one PDMS. For simplicity, we call *colluding nodes* the leaking PDMSs (passive attack) or corrupted PDMSs (active attack) controlled by the same attacker. It is important to note that the worst case attack is represented by the *maximum number of colluding nodes controlled by a single "attacker"*. The remaining question is thus: how many colluding nodes could an attacker control?

Collusion extent. Creating a large group of colluding nodes presents two main difficulties: (i) the need to remain indistinguishable from honest nodes as detected malicious nodes can easily be excluded [7] and (ii) possessing the necessary equipment and/or sufficient knowledge to perform advanced attacks on the PDMS. Since each PDMS is associated with a real individual (e.g., by only delivering the device to real users proving their identity), collusion between individuals remains possible but can hardly scale without being minimally advertised, hence making them distinguishable and breaking their cover. Besides, enough individual attackers must be willing to collaborate (despite mutual distrust) to create a large network of compromised nodes. Thus, wide collusion is extremely difficult to build since it calls for significant organization between a large number of users, which, in practice, requires an extremely powerful attacker as well as extreme discretion. Nonetheless, we consider in this paper that a powerful attacker controls a rather large number of colluding nodes. Although worrisome, such a situation does not prevent our system from functioning and from completing our objective. A calibration is however necessary using a, potentially overestimated, maximum number of colluding nodes controlled by an attacker.

3.2 Lower-bound on leakage and Problem Formulation

In this sub-section, we do not further consider the idealistic Tamper-proof threat model which has an obvious lower-bound on leakage of zero. Indeed, in this threat model used as a baseline, PDMSs nodes are fully honest; we can therefore expect no leakage as soon as we properly protect the communications between the actors and the targets.

Potentially exposed data. Data leakages can happen for *data-at-rest*, by leaking (i) one or several concept indexes; and *data-in-use*, i.e., data or metadata exchanged or manipulated during the query execution by leaking (ii) some *TIPs*, (iii) some local results, or even worse, (iv) some *TIPs* linked with the corresponding local results, called *full association* in the rest of the paper. Note that we do not consider the direct leakage of user's data since each PDMS is locally secure (see Assumption 3), and for nodes controlled by the attacker, there is no benefit in performing a "self-attack".

Abstract query execution: targeters and aggregators. We can distinguish two phases in the query execution, whatever the threat model, the protocol or the techniques used. Each phase should be conducted by a distinct set of nodes (otherwise, as with Naive execution, the leakage risk is increased). In an abstract way, we distinguish between (1) targeters that find targets thanks to concept indexes and transmit the local query to the sampled ones and (2) aggregators that aggregate the local results, working on clear-text data (see Section 1).

Unavoidable leakage. We consider the worst case attack, i.e., an attacker who masters up to C leaking or corrupted nodes (with $C < N$, N being the total number of system nodes), and acting as covert adversaries (i.e., indistinguishable from honest nodes). As already mentioned, there is no other computing element in the architecture (trusted server for instance), and thus, only PDMSs can be selected as query actors. Thus, leaking or corrupted nodes may be chosen as targeters, as aggregators or in both groups. In this configuration, data leakages are unavoidable, and thus, our only leeway is *to minimize the leakage risk*, and if a leakage does occur, *to minimize it's impact*.

Random actor selection. How can we minimize the risk of selecting leaking or corrupted nodes, if we do not have any clue on nodes honesty (covert adversaries)? Actually, the best strategy is to select query actors randomly, thus leading to a fraction of, on average, C/N leaking or corrupted nodes in targeters and aggregators. A consequence of randomization [50, 7] is that it leads to *guarantees in average*. This means that the lower-bound on leakage discussed below represents an expected average value over a number of queries, and that some small random variations can be expected between queries.

Lower-bound on leakage. Since the targeters must contact the selected targets accessing their *TIPs* and the aggregators must manipulate the local results in the clear, it is then obvious that the lower-bound on leakage is on average a fraction C/N (i.e., strictly proportional with the percentage of colluding nodes) of the IP addresses and local results of the selected targets. We can however expect to protect with near-certainty, i.e., with a very high and adjustable probability (similar to the protocols for communication anonymization such as Tor [50]) the rest of the sensitive data (concept indexes and the full association between *TIPs* and local results) since this depends on the way the concept indexes are

stored and on how the distribution of tasks to actors is done (i.e., minimization of the leakage impact).

Problem formulation. The problem is then to provide protocols, for managing both data-at-rest (i.e., concept index insertion protocols) and data-in-use (i.e., query protocols), which reach the expected lower-bound on leakage with near-certainty. The computation of this probability is based on a, potentially overestimated, maximum number of colluding nodes controlled by an attacker (see Section 7.2). Obviously, security has a non-negligible cost and thus, we consider as a second objective the protocols efficiency[2].

Approach. We study in the next three sections the three incrementally difficult threat models. To further ease the reading, we introduce the different protocols in the same manner: (i) we present the additional security requirements and corresponding security techniques imposed by the threat model; (ii) we describe the insertion of a Node's profile and continue with the query processing by detailing (iii) the new required actors that enforce the security requirements and the related efficiency considerations and by giving (iv) a step by step description of the protocol followed by a discussion of the different choices made; and finally, (v) we conduct a security analysis highlighting the parameters leveraged to reach the lower-bound on leakage and discuss their limits.

4 Tamper-proof Threat Model and DISPERS^{H}

This model assumes that the PDMS security cannot be compromised, leaving only the communications open to attacks.

4.1 Security Requirements

By listening to the communications, an attacker can infer information based on the data itself (i.e., the content) or on the metadata (i.e., who is communicating with whom, when, after whom, etc.) leading to the following requirement:

> **Requirement 1: Hidden communications.** All sensitive exchanged data and metadata should be protected such that an attacker cannot gain knowledge by spying on a subset of nodes (see Assumption 1).

To enforce this requirement, we rely on: (i) *encryption* and (ii) *anonymization*.

The choice of the correct encryption scheme (see Appendix A) is context dependent and is discussed in each protocol, but, in any case, every encrypted communication includes a hash of the message to *protect its integrity*. Note that for hybrid encryption, we follow the current communication security standard and rely on the TLS protocol to establish secure channels providing both the message integrity and its encryption, based on the public key certified by an authority according to Assumption 2.

Considering Assumption 1, we provide anonymity by dynamically introducing one or more *proxies* between nodes whose communications should not be linked. By "dynamically", we mean that these proxies are chosen *during* the distributed computation and changed at *every* computation. Indeed, by doing so an attacker does not know which nodes to listen to. Also, evidently, the more proxies there are, the more difficult it is for an attacker to successfully uncover a node (see the detailed analysis provided in Section 4.5). We propose to use a custom solution based on onion routing, largely inspired by Tor [21]. This is preferred to using directly Tor for several reasons: (i) The anonymous routing is achieved using the system nodes (proxies), so without relying on an external anonymizer, which would introduce an additional (strong) trust assumption and would further complexify the system. Existing work [11] shows that de-anonymizing users on Tor is possible, especially when interconnecting DHT-based applications like BitTorrent with Tor. (ii) Our solution allows tuning the security level (i.e., the number of proxies) based on the expected attack level (i.e., the maximum number of systems nodes that can be spied by the attacker), which is not possible with an external system like Tor. (iii) Using an external anonymizer would make difficult the evaluation of the anonymization cost. (iv) The Tamper-proof threat model allows proposing a more efficient protocol than Tor onion-routing (see Appendix A) given that proxies are trustworthy and cannot access the data being forwarded. In such a case, we propose that the sender selects only the first proxy which then selects the following proxy and so forth until the message is eventually delivered to the receiver. We call those proxies *basic proxies*. The advantage of basic proxies is that the asymmetric encryption overhead is better distributed, putting less stress on the sender node. Basic proxies cannot be used for the other threat models.

4.2 Insertion in the DHT

Inserting a profile in the DHT requires contacting the CIs indexing the different concepts composing the profile. As we cannot hide the CIs (how could we contact them?), we can posit that an attacker will acquire their IP address and intercept their communications. To protect the nodes and their profiles we thus encrypt and anonymize the communications with the CIs, as illustrated in Fig. 4, leading to two new roles: The *Node Proxy* (NP) proceeds to the insertion of a concept in lieu of the node: it locates the corresponding CI in the DHT and then asks it to store the concept (and *TIP*);

[2] Issues related to statistical databases (e.g., inferences from results [71], authorized queries, query replay) or to network security (e.g., message drop/delay, routing table poisoning [72]) are complementary to this work and fall outside its scope (see Sections 8 and 9).

the *Proxy* (P) acts as a relay forwarding a communication to the next P or to NP. The insertion protocol is as following:

Protocol 2. Insertion protocol

1. For each concept composing the profile, the node (N) randomly selects a Proxy (P) and asks it to forward, as many times as required, the insertion query.
2. The last proxy, which acts as a Node Proxy (NP) makes a DHT lookup operation to contact the CI. NP provides the concept, its own IP address and certificate without encrypting them (they are not sensitive).
3. The CI establishes a secure communication channel with NP. NP finally inserts the sensitive couple (concept, TIP).

4.3 New Roles for Query Processing (w.r.t. Requirement 1)

As for the profile insertion in the DHT, we must protect the Targets during a query execution (hiding their *TIPs*) because they match a given profile, potentially known by a malicious querier. We thus place basic proxies "before" and "after" the Targets: The *Before Proxies* (BPs), resp. *After Proxies* (APs), forward the communications going to (resp. from) a Target to hide its *TIP* from an attacker eavesdropping the actor sending the Local Query (resp. receiving local results).

To avoid disclosing Targets, complying with Requirement 1, we must encrypt their incoming communications. This is not obvious since Targets are discovered dynamically and, thus, BPs cannot know their public keys. The only solution is to store in the CIs, in addition to *TIP*, either a symmetric key k_{sym}, different for each (target, concept) couple, or the Target public key k_{pub}. Considering that symmetric encryption is significantly more efficient, we select the former. The actors then choose, for each Target, one of the available k_{sym} (if there are several concepts), encrypt the Local Query with k_{sym}, accompany it with $\texttt{hash}(k_{sym})$ and *TIP*, and send them to a proxy. The Target knows, through $\texttt{hash}(k_{sym})$, which k_{sym} must be used to decrypt the Local Query, and later, to encrypt its local result. The actors reuse k_{sym} to decrypt the local results.

If Q contacts all the sampled Ts (through BPs), we face two issues: (i) from a security viewpoint, Q knows all the first level BPs, thus increasing the probability of discovering Targets (the same is true for APs); (ii) from an efficiency viewpoint, Q becomes a bottleneck. To solve both, we propose to add a set of supporting nodes called *Workers* (Ws). Ws supplant Q to (i) transfer the Local Query to a subset of the Targets and (ii) aggregate their local results[3] before sending partial results back to the Querier.

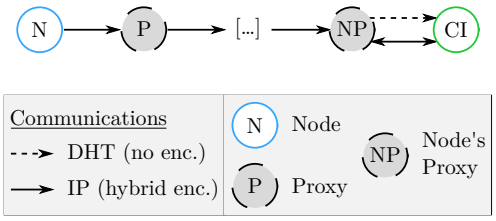

Fig. 4: Insertion of a concept in the DHT

Unfortunately, if the Ws are selected in Q's neighborhood or Q's cache (each node maintains a local cache of recently contacted nodes), an attacker could deduce which nodes to spy by analyzing past communications, or influencing the local cache by issuing specific queries. Thus, we propose to delegate this selection to an *Actor Selector* chosen randomly by Q by making a lookup in the DHT at a random place (see Section 4.4). Randomly relocating the selection of actors allows: (i) distributing the potential leaks in a different region for each computation and (ii) balancing the load, improving the overall performance.

As shown in Fig. 5, these limited additions to the Naive protocol allows securing query computations in the Tamper-proof model. Encrypting the communications plus adding proxies and workers chosen randomly by the AS, allows to hide the sensitive nodes and ensure the security guarantees we aim for. Additionally, the Workers optimize the execution flow and distribute the load put initially on the Querier.

4.4 Detailed Protocol

The optimized Tamper proof compliant protocol DISPERS$^{\text{H}}$ is given below (see also Figure 5). Note that steps 1 and 3 as well as 2 and 4 are done in parallel.

Protocol 3. DISPERS$^{\text{H}}$: Tamper-proof compliant protocol

1. The Querier (Q) generates a random number RND_Q and hashes it to obtain a location in the DHT virtual space. Q contacts through the DHT, in clear, the node managing this location: the Actor Selector (AS).
2. The AS checks Q's certificate to ensure that Q is a genuine PDMS, randomly selects the Workers (W) in its local cache, and sends the list to Q (hybrid encryption).
3. Q checks AS's certificate and looks up in the DHT, in clear, the Concept Indexers (CIs) indexing the different concepts composing the Target Profile (*tp*).
4. Each CI and Q mutually check their certificates. The CIs send to Q the encrypted list of $(TIP,\ k_{sym})$ associated to the requested concept (hybrid encryption).
5. Q applies the *tp* on the lists to find the Targets (T). It then selects a random sample and evenly splits the sample in $|\text{W}|$ sets. It finally sends to each W the Local Query (*lq*), the Aggregate Query (*aq*) and a set of sampled $(TIP,\ k_{sym})$.

[3] This is possible with distributive aggregation expression, i.e., the aggregate computation can be distributed on several data processors.

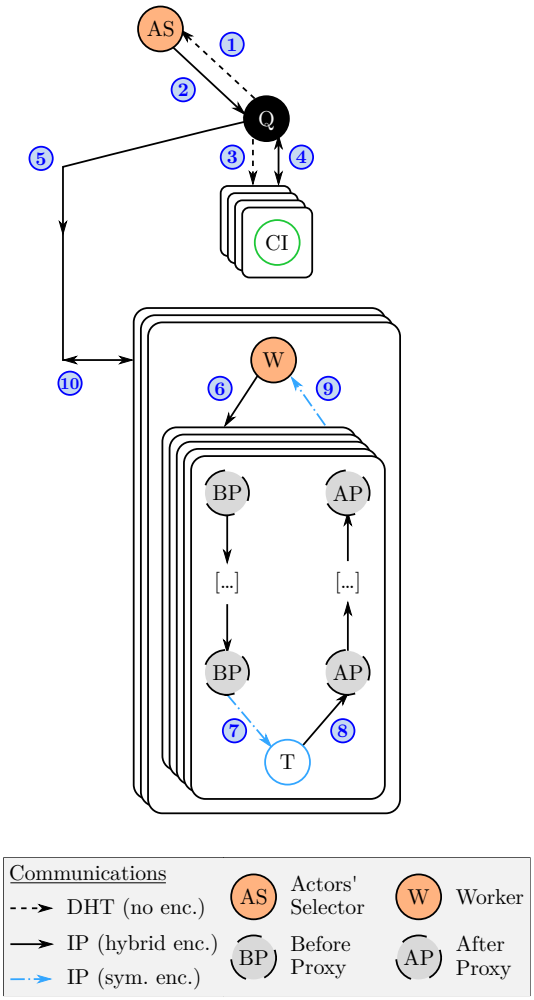

Fig. 5: DISPERS^{H}: Tamper proof compliant protocol

6. Each W encrypts for each T, using one of the available k_{sym}, the *lq* and W's IP address. These messages are sent (encrypted) to randomly selected Before Proxies (BP).
7. The BPs forward to the Ts the encrypted data. The communication between each last BP and T is encrypted using k_{sym} — this encryption being performed at step 6.
8. The Ts apply the *lq* on their data and encrypt their local result using k_{sym}. They finally randomly select After Proxies (AP) and ask them to forward their encrypted results to their respective W. Note that, unlike step 2 or 4, there is no need to check that the Ws are genuine PDMSs since, if not, they could not have provided a correct k_{sym} (i.e., known by T).
9. The APs forward the local results to the Ws.
10. The Ws decrypt the local results, apply *aq* and send its results to Q.
11. Q applies *aq* on the partial results to obtain the final answer and thus ends the query.

Table 3 provides a summary of the analysis of the protocol, mainly with respect to Requirement 1. For each step, we indicate if communications are encrypted, using symmetric (green) or hybrid encryption (yellow) and if the step can be parallelized (green) for improved performance. We indicate the (most, if multiple) sensitive information we protect through encryption.

As we can see, all communications are encrypted except in Step 1 and 3 (DHT lookups). The rationale for Step 3 is that the concepts (potentially sensitive) cannot be protected when the owner of the Querier node is malicious. All steps are also parallelized, except for the selection of Ws (step 1) and the application of *tp* (step 3). Both are not expensive tasks and would not benefit from a parallelization.

4.5 Security Analysis

Since, in the Tamper-proof model, the data manipulated by PDMSs are not at risk and since every sensitive communication is encrypted, an attacker may only deduce some Target IPs (*TIPs*) from the metadata — which she can then associate with a profile as she potentially formulates the query. Before and After Proxies help anonymize the communications between Ws and Ts, but how do we calibrate this number of proxies?

Let us quantify the probability of leaking one or more *TIPs*, $\mathbb{P}_{T^+}$, when an attacker spies the communications of C nodes in a network of N nodes, using p BPs and p APs and $|\mathrm{T}|$ Targets. We denote c, the ratio of spied nodes ($c = \frac{C}{N}$).

$$\mathbb{P}_{T^+} = 1 - \left(1 - 2c \times \left(\sum_{i=0}^{\lfloor p/2 \rfloor} \binom{p-i}{i} \cdot c^{\,p-i} \cdot (1-c)^i\right) - c \times \left(\sum_{i=0}^{\lfloor p/2 \rfloor} \binom{p-i}{i} \cdot c^{\,p-i} \cdot (1-c)^i\right)^2\right)^{|T|} \quad (1)$$

Indeed, an attacker can deduce the *TIP* of a Target T iff: (i) he is able to deduce (given the quantity of sent messages) that some spied nodes are Worker nodes; (ii) he is, by

STEP	ENCRYPTION	PARALLELISM
1: (Q→AS)	No	Yes, automatic (DHT)
2: (AS→Q)	Hyb.: $\{(IP_W, Cert_W)\}$	No, light task
3: (Q→CI)	No	Yes, automatic (DHT)
4: (CI→Q)	Hyb.: $\{(TIP, k_{sym})\}$	Yes, if multiple CIs
5: (Q→W)	Hyb.: $\{(TIP, k_{sym})\}$	No, Q applies *tp*
6: (W→BPs)	Hyb.: (TIP, k_{sym})	Yes, multiple Workers
7: (BP→T)	Sym. (k_{sym}): *lq*	Yes, one BP per Target
8: (T→AP)	Hyb.: *TIP*	Yes, multiple Targets
9: (APs→W)	Sym. (k_{sym}): local res.	Yes, one AP per Target
10: (W→Q)	Hyb.: partial res.	Yes, multiple Workers
11: (Q)	No communication	No, light task

Table 3: DISPERS^{H} protocol analysis

chance, (a) spying the Worker W targeting T (thus obtaining the address of the first BP); (b) spying the last BP between W and T (thus obtaining T's *TIP*); and (c) spying BPs between W and T such that no two consecutive BPs are not spied (thus being able to link the whole chain of BPs). The same observations can be done with APs (the first AP must be spied). For instance, with 7 proxies, 10^3 Targets and 1% spied nodes, $\mathbb{P}_{T^+} < 10^{-6}$. Such a protection level is more than acceptable since a prerequisite is that a Worker is identified, which is not trivial with queries running in parallel.

4.6 Conclusion

The proposed protocols address the first requirement using encryption and anonymization and parallelize the computations when possible. The security analysis shows that we can reach the lower-bound on leakage (no leakage) with a probability $\mathbb{P}_{T^+}$ which can be tuned to be as small as required, i.e., with near certainty. Considering tamper-resistant PDMS nodes can be, however, too restrictive. If an attacker were to break a single PDMS then all the sensitive data would leak: this node only has to initiate a query to observe the full list of Targets and partially aggregated results. We thus address next, our second, more invasive, passive attack threat model.

5 Passive Attack Threat Model and DISPERS^{HR}

In the Passive attack threat model, in addition to spying on the communications, an attacker can uncover, without being able to modify, the data normally protected by the PDMS.

5.1 Security Requirements

As mentioned in Section 3.2, when an attacker controls a set of leaking nodes, there is an unavoidable data leakage and thus, our objective is to minimize its impact and reach the lower-bound on leakage, while keeping the security costs low. We thus propose the following requirement:

> **Requirement 2: Random dispersion of data.** Data-at-rest (i.e., the distributed concept index) and data-in-use (i.e., the data exchanged during query execution) must be ***dispersed*** on nodes chosen ***randomly***.

Indeed, the best achievable protection is obtained with random actor selection (CIs and query actors are randomly selected and that selection cannot be influenced by the attacker). Then, through dispersion, we minimize the information each actor receives and, de facto, limit the impact of a leak. We remind that the users' own data do not need dispersion since there is no gain in performing a self-attack. We make use of the following security techniques:

Imposed, uniformly distributed node location in the DHT. Imposing the location of nodes in the DHT ensures a uniform density of the leaking nodes since an attacker cannot influence its location. This can be easily achieved by using the hash of the node's public key as its identifier in the DHT. Indeed, hash functions produce uniform distributions, and public keys are unique and cannot be influenced by the PDMS owner. Coupled with a random assignation of the actors and the CIs (e.g., the hash of the concept for the CIs), we obtain a random selection of the query actors.

Shamir's Secret Sharing Scheme (SSSS) for random dispersion of data-at-rest. By using SSSS (see Appendix A) in combination with the random assignment of the CIs, we can safely store the concept indexes in the DHT.

Task compartmentalization for data-in-use random dispersion. By compartmentalizing a task in several independent sub-tasks, coupled with the random assignment of the actors, we can reduce the data transferred to the actors to the minimum required for each sub-task, effectively dispersing them on several actors. A task should be compartmentalized iff each sub-task requires less data to be performed, so that it minimizes the leakage impact if any of the actors is leaking.

Pseudonymization of data-in-use. To further reduce the impact of a leakage of data-in-use, we rely on pseudonymization in case an actor has access to more information than it needs. $\texttt{Pseudonym}(\texttt{x})$ is denoted $\overline{x}$ and allows "hiding", when applicable, the concepts of the Target Profile, the Aggregate Query, or the IP addresses of the Targets. By replacing a concept c_i in the Target Profile with $\overline{c_i}$, we can apply it on the lists sent by the CIs without revealing any of the concepts. For instance, the pseudonymized profile $\overline{tp} = (a \wedge b)$ replaces $tp = (\texttt{location}|\texttt{city}|\mathit{Lyon}) \wedge (\texttt{profession}|\mathit{none})$. Similarly, we replace the "subject" of the Aggregate Query with a pseudonym. Instead of computing *aq*: "average of the ratings", we can compute the $\overline{aq}$: "average of x" hence hiding the nature of the local results (to some extent). Finally, swapping the Targets' IP addresses (*TIP*) with pseudonyms $\overline{TIP}$ can greatly reduce data-in-use leakage as detailed in Section 5.3. Note that, depending on the computation performed, one may employ dedicated obfuscation methods to further protect the local results (e.g., local differential privacy [77]). Our purpose here is to be generic and we thus leave these optimizations to future work.

Sub-task parallelization. Lastly, parallelizing the sub-tasks, i.e., using several nodes to compute a sub-task, further reduces the amount of data accessed by each actor and thus the potential leakage, and speed up the overall execution.

5.2 Insertion in the DHT

As explained earlier, proxies can now leak the data they manipulate, thus requiring onion routing to operate blindly (see

Appendix A). Furthermore, even if the Node Proxy is leaking, it has, thanks to SSSS, only access to a share of the IP address of the Node inserting the concept in the DHT.

Overall, the protocol remains the same as Protocol 2: there are now as many insertions as there are shares due to SSSS (see Fig. 6) and the communications between the proxies follow the *onion routing* protocol. The CI responsible for storing the share j of concept c_i is the node responsible for the key computed for instance as $\texttt{hash}(\text{`['} \mid c_i \mid \text{`]'} \mid j)$ to randomly disperse the shares on different CIs (thanks to the uniform distribution of cryptographic hash functions). Hence, by splitting their *TIP* into *shares* assigned to different CIs, the probability to leak the *TIPs* for a concept decreases exponentially with the number of shares (see Section 5.5).

5.3 New Roles for Query Processing (w.r.t. Requirement 2)

Analyzing the Querier and Workers roles in Protocol 3, we identify four different tasks: (i) Q contacts the CIs, (ii) Q applies *tp* and transmits the sampled Targets to Ws, (iii) Ws transfer the Local Query to the resulting (sampled) Targets, and (iv) W receive and aggregate the local/partial results.

To fulfill the *Random dispersion of data* requirement, we restrict the role of Q to contacting the CIs and introduce four new roles resulting in the compartmentalization of the tasks (ii), (iii) and (iv) respectively leading to PS, TF and (F)DA. For each sub-task, the data-in-use transmitted to the new actor nodes is minimized (minimal knowledge) and pseudonymized whenever possible.

The *Profile Samplers* (PSs) (i) reconstruct $\overline{TIP}$ and apply $\overline{tp}$ to determine the Targets, (ii) select a sample of Targets and (iii) transfer the sample data to a *Target Finder*. Thus, the PSs determine and sample the Targets without knowing any IP address or the Target Profile.

The *Target Finders* (TFs) (i) reconstruct *TIP* only for the sampled Targets, (ii) transfer, anonymously via proxies, the encrypted Local Query to the sampled Targets. Thus, the TFs communicate with the sampled Targets without knowing their pseudonyms nor the Target Profile.

The *Data Aggregators* (DAs) aggregate the local results using the pseudonymized Aggregate Query.

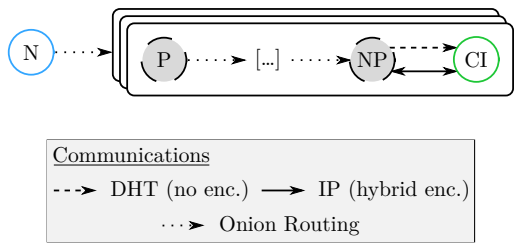

Fig. 6: Insertion of a concept in the DHT

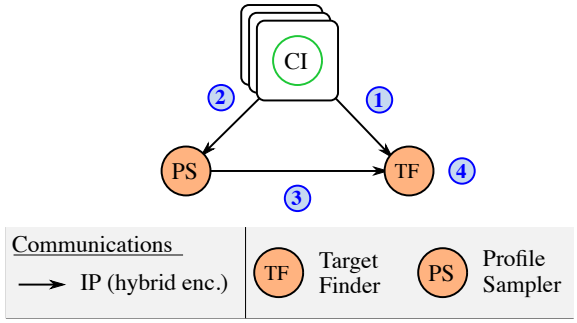

Fig. 7: PS/TF interactions

For each role, several nodes can be selected to parallelize the execution of the sub-task. Having several DAs requires a final aggregation performed by the *Final Data Aggregator* (FDA) which then transfers the final result to the Querier.

Fig. 7 illustrates how the task and information distribution is performed to minimize their knowledge, assuming a single PS and TF (see Section 5.4 for multiple ones).

1. Each CI sends to the TF each share of (*TIP*, k_{sym}) encrypted with a "temporary" symmetric key, k_{tmp}, generated on-the-fly for each share. k_{sym} is used to encrypt the communications with the Target (see Section 4.3), while k_{tmp} is used to hide from TF the (*TIP*, k_{sym}) of the Targets that are not selected by the sampling.
2. Each CI sends to the PS the concept pseudonym ($\overline{c_i}$), and for each node having c_i, the share of $\overline{TIP}$ and k_{tmp}.
3. The PS reconstructs the nodes' $\overline{TIP}$ (see Section 5.4), applies $\overline{tp}$ on $\overline{TIP}$s, samples the resulting Targets and finally sends to the TF the set of k_{tmp} of the sampled Targets (one per share of sampled Target).
4. The TF decrypts the shares of (*TIP*, k_{sym}) of the sampled Targets using the adequate k_{tmp}s, reconstructs the (*TIP*, k_{sym}) and finally encrypts (with k_{sym}) and transfers them (anonymously) the Local Query and other metadata.

The PS thus only knows the pseudonyms ($\overline{TIP}$) of the nodes possessing any of the concepts in *tp* — without knowing which — and the TF knows the *TIPs* of only the sampled Targets without knowing *tp*. Providing them less information would make them unable to perform their respective tasks, hence showing that this task distribution is optimal.

5.4 Detailed Protocol

Fig. 8 gives an overview of DISPERS^{HR}, the passive attack compliant protocol. We voluntarily omit the proxies to improve readability and to focus on the new flow. Details are addressed just after.

Protocol 4. DISPERS^{HR}: Passive attack compliant protocol

1. Q generates RND_Q and looks up in the DHT the AS, the node managing the location $\texttt{hash}(RND_Q)$. The AS checks Q's certificate to ensure that Q is a genuine PDMS, selects randomly a list of actors in its local cache, and sends this list to Q (hybrid encryption).
2. Q transmits (hybrid encryption) the required metadata to each actor: (i) the $\overline{tp}$ and list of TFs to the PSs, (ii) the lq and list of DAs to the TFs, (iii) the $\overline{aq}$ to the DAs/FDA.
3. Q looks up in the DHT each CI managing a share of each concept c_i of the tp. Then, Q provides (hybrid encryption) to each CI the pseudonym $\overline{c_i}$ and the list of PSs and TFs.
4. Each CI checks Q's certificate, and forwards to the PS, the $\overline{c_i}$ and, for each node having c_i, k_{tmp}, and a share of $\overline{TIP}$; and to the TF, for each node having c_i, a share of (TIP, k_{sym}) encrypted with k_{tmp}.
5. Each PS reconstructs the $\overline{TIP}$s, applies $\overline{tp}$, samples the resulting Targets and finally sends to its corresponding TF the set of k_{tmp} of the sampled Targets.
6. Each TF decrypts the shares of (TIP, k_{sym}) of the sampled Targets using the adequate $k_{tmp}s$, reconstructs the (TIP, k_{sym}) and finally encrypts with k_{sym} and transfers to the Targets (onion routing) the lq and the list of DAs.
7. Each Target deciphers the lq and list of DAs using k_{sym}, applies lq, chooses randomly a DA and forwards it anonymously (onion routing) its local result.
8. The DAs apply $\overline{aq}$ on the local results, and send (hybrid encryption) the partial results to the FDA.
9. The FDA applies $\overline{aq}$ to compute the final result and sends it to Q (hybrid encryption).

Selecting the actors. A correct selection of the actors is crucial as the leakage is linked to how many of them are leaking. As we have established in Section 3.2, that selection must be *random* to minimize, on average, that number. We used the Actor Selector as in the Tamper-proof model to select actors in randomized regions of the DHT, thus distributing the load and the leakages while ensuring a uniform and random selection of actors.

Multiple (PS, TF). Having multiple instances of (PS, TF) couples (each PS communicate with its corresponding TF) raises two issues: (i) how can the CIs send the shares belonging to the same node to the same actor and (ii) how can an actor know which shares go together?

To address (i), the nodes store a *selector* called *sel* with their share entries in the DHT. *sel* is identical for all entries. The CIs leveraged *sel* to determine to which actor they should transfer it by using a modulo. *sel* should be chosen so as to favor collisions and avoid creating a second, insecure, pseudonym. DISPERS uses a hash of the node's public key concatenated with a random value (to reduce the attacker knowledge): $sel = \texttt{hash}(k_{pub}\,|\,RND) \mod |\text{PS}|$.

To address (ii), considering that SSSS does not indicate if a set of shares are related to the same secret, we associate a "marker" to each set, called a *Concept-Target Identifier* $(CTID)$, to signal this relationship. As there are possibly many shares for as many different secrets, this marker is unique per secret and should depend on the Target and the concept to avoid mixing shares that do not belong together. Once more, we use a hash of the node's public key concatenated with the concept and a random value (to avoid rainbow table attack): $CTID = \texttt{hash}(k_{pub}\,|\,c_i\,|\,RND)$.

Summary of the data stored at a CI. To summarize, a CI responsible for the share j of concept c_i stores the tuple $(CTID, sel, (TIP, k_{sym})_j, \overline{TIP}_j)$ for each node having c_i in its profile where: (i) $CTID$ is a Concept-Target Identifier, unique for the couple (concept, node) but equal for all its shares; (ii) sel is a selector ensuring that all the shares of the same node (whatever the concept) are sent to the same PS and TF; (iii) $(TIP, k_{sym})_j$ is a share of the node IP address

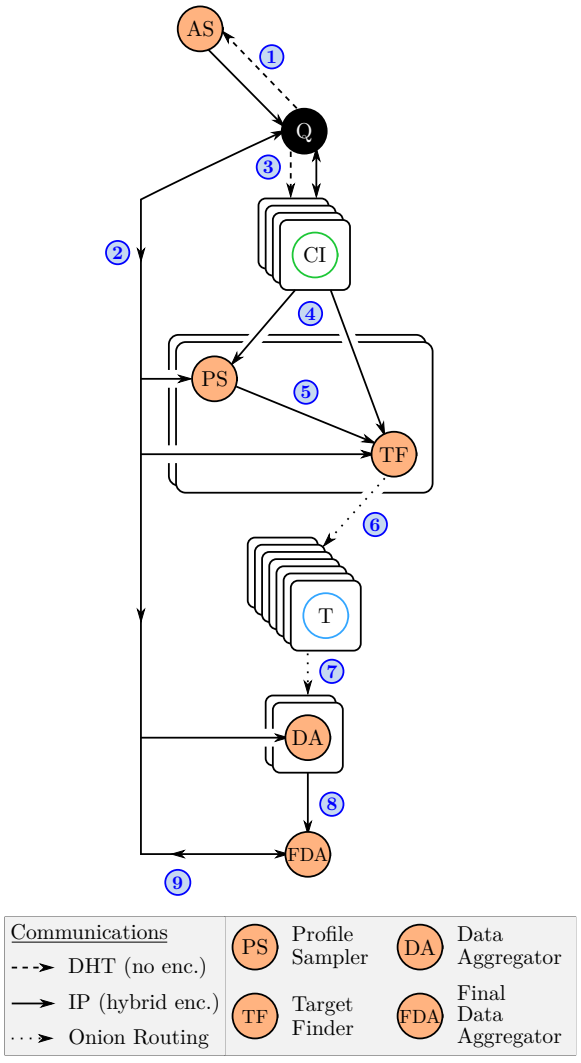

Fig. 8: Overview of the DISPERS^{HR} protocol

and of the symmetric key, unique for the couple (concept, node); and (iv) $\overline{TIP}_j$ is a share of the node pseudonym.

Table 4 shows that the metadata (in green) accessed by each role are minimized: no role except Q has access to more metadata than strictly necessary. This is important to avoid *opportunistic attacks*, i.e., when the attacker does not control the querier node and thus, has no access to the query. More importantly, each role receives only the necessary data (in red) to perform its task. For instance, TFs receive *lq* and some Targets *TIPs* because they need to send the former to the latter. The next section quantifies the impact an attacker mastering different actors would have. In particular, we study how combined leakages would help deduce concept indexes, *TIPs*, local results or full associations.

5.5 Security Analysis

As mentioned in Section 3.2, we must consider the leakage of (i) one or several concept indexes; (ii) some *TIPs*, (iii) some local results, and (iv) their full association.

Concept index leakage. A concept index for a concept c_i, protected by SSSS with n shares and a threshold of t, is disclosed if an attacker controls at least t nodes storing shares of c_i's index. Thus, the probability of leaking one or more concept index is equal to 1 minus the probability of not leaking any concept, i.e., having for each concept at most $t-1$ leaking concept indexers (see equation 2 with $|c_i|$ concepts indexes in the system, c is the ratio of leaking nodes). Using this formula, we can easily calibrate the system to obtain a very low probability. For instance, with 11 shares and a threshold of 8 (increasing redundancy and making the system robust to failures, see Section 7), the probability of leaking one or more concepts when the attacker controls 1% of the nodes, is around 10^{-7} with a total of 10^5 concepts.

$$\mathbb{P}_I = 1 - \left(\sum_{i=0}^{t-1} \binom{n}{i} \times c^i \times (1-c)^{n-i} \right)^{|c_i|} \quad (2)$$

Meta data \ Role	Q	CI	PS	TF	T	DA	FDA
Concept (c)	×	×					
Pseudonymized c_i ($\overline{c_i}$)	×	×	×				
Target Profile (tp)	×						
Pseudonymized tp ($\overline{tp}$)	×		×				
Local Query (lq)	×			×	×		
Aggregate Query (aq)	×						
Pseudonymized aq ($\overline{aq}$)	×					×	×
Node pseudonym ($\overline{TIP}$)			×				
TIP of sampled Targets				×			
Local Results					×	×	
Partial Results						×	×
Final Result	×						×

Table 4: Summary of the data accessed by each role.

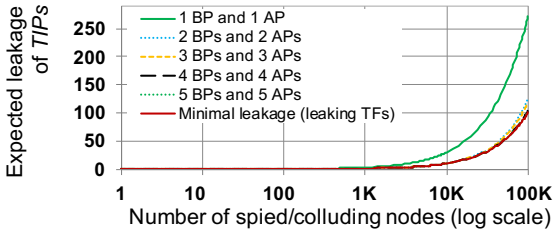

Fig. 9: Effect of proxies on *TIPs* leakage

Sampled Target address leakage. There are three *TIP* leakage scenarios: (a) a TF node is leaking; (b) a "chain" of BPs is spied or leaking (same reasoning as in Section 4.5), see equation 3 with $p > 1$ BPs; (c) a "chain" of APs is spied or leaking, see equation 4 with $p > 1$ APs. The probability of leaking one or more *TIP* is then given by equation 5.

$$\mathbb{P}_{BP} = \sum_{i=0}^{\lfloor p/2 \rfloor} \binom{p-1-i}{i} \times c^{\,p-i} \times (1-c)^i \quad (3)$$

$$\mathbb{P}_{AP} = \sum_{i=0}^{\lfloor (p+1)/2 \rfloor} \binom{p+1-i}{i} \times c^{\,p+1-i} \times (1-c)^i \quad (4)$$

$$\mathbb{P}_{T^+} = 1 - ((1-c) \times (1-\mathbb{P}_{BP}) \times (1-\mathbb{P}_{AP}))^{|T|} \quad (5)$$

Scenario (a) leads to an *unavoidable* leakage (on average) of $T \times c$ Target IPs— given our fully-distributed setup and to the indistinguishability of leaking nodes. The objective is thus to minimize the extra leakage of scenario (b) and (c), which depends on the length of the "chains".

Fig. 9 shows the impact of the addition of BPs and APs (same number) on the expected *TIPs* leakage. With only one proxy of each type both leakages are equal to the one of scenario (a): each proxy is either leaking or not thus leading to three chances for a *TIP* to be leaking: a leaking TF, BP or AP. Having two or more proxies severely reduces it (a leaking chain is required). However, extending the length of the chains rapidly yields little to no benefits, especially when compared to the, unavoidable, leakage of scenario (a).

Local result leakage. Local results leak iff one or more DAs are leaking. This leakage, on average, $T \times c$ local results, cannot be reduced since DAs work on clear text and leaking DAs are indistinguishable from honest ones. Fortunately, local results alone do not bring many insights to the attacker, especially if *lq* can be restricted in some ways (e.g., *lq* should not return identifying results). Considering working on ciphertext is part of our future work (see preliminary results in [45]).

Full association leakage. The full association leakage requires that (i) the first AP is spied or leaking; (ii) no two consecutive APs are honest (and not spied) between the first

AP and the DA; (iii) the DA is leaking. (i) and (ii) allows obtaining the *TIP* while (iii) allows obtaining the local result and linking it with the *TIP*. Equation 6 gives the probability of leaking one or more full association.

$$\mathbb{P}_F = 1 - ((1 - \mathbb{P}_{AP}) + (1 - c) - (1 - \mathbb{P}_{AP}) \times (1 - c))^{|T|} \quad (6)$$

As we can see on Fig. 10, we can reduce this probability, as much as desired, by increasing the number of APs. For instance, with 7 APs and 10^4 colluding nodes (1%), the probability is inferior to 10^{-6}. Note that adding APs is a costly mechanism: each Target must perform as many asymmetric encryption operations as there are APs and each AP a decryption. However, all these operations are done in parallel and should have a marginal impact on the execution latency.

5.6 Conclusion

By compartmentalizing the query execution on multiple actors, hiding the data with pseudonyms or SSSS, we managed to isolate the sensitive information such that each actor has a very limited view on the global data processing. Moreover, by controlling the number of SSSS shares (n and t), and the number of BPs and APs, we can influence $\mathbb{P}_{T^+}$ and $\mathbb{P}_F$ such that we reach the lower-bound on leakage with near certainty. However, considering corrupted nodes changes the game: a corrupted Querier can choose colluding nodes as actors thus leaking all the sensitive data. This is studied in the next section.

6 Active Attack Threat Model and DISPERS^{HRC}

In the Active attack threat model, we assume that colluding attackers have complete control over their own PDMSs: they can alter the code integrity and forge fake information. Hence, although the overall data insertion and query protocols remain the same as in the passive attack model, additional attestations have to be carefully generated and then verified by the concerned nodes to **maintain the same level of security**. This section is based on our previous work [40] that we extend in a more comprehensive and generic way.

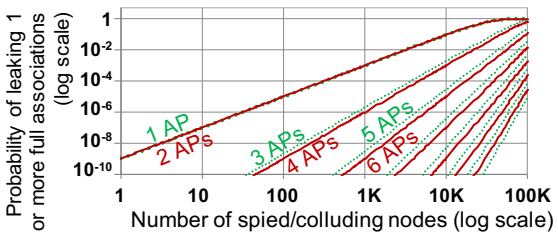

Fig. 10: Effect of APs on full association leakage

6.1 Security Requirements

The main consequence of this threat model is that nodes, especially those providing sensitive data, cannot trust any information they receive even if it is signed by a, potentially corrupted, PDMS. A notable exception is the node's certificate which is the security root (see Assumption 2). We identify three types of attacks: (i) *impersonation*: when a corrupted node impersonates another node (e.g., impersonating a CI to reveal some concept profile); (ii) *list of corrupted actors*: when a corrupted Querier produces a list of actors containing mostly (if not only) corrupted nodes; and (iii) *query manipulation*: when corrupted nodes bypass their query budget or alter the query parameters during the execution (e.g., a corrupted TF changing *lq* to make a Target reveal itself). The following requirement addresses these issues:

> **Requirement 3: Collaborative proof.** Any "information" that leads (directly or indirectly) to the transmission of sensitive data must be (i) attested and (if applicable) generated through a collaborative process involving honest nodes, and (ii) the participation of honest nodes in this process must guarantee its correctness, i.e., a single honest node has the power to invalidate it.

A collaborative process prevents an attack from a single corrupted node. By additionally forcing the presence of honest nodes and giving them the power to invalidate the produced information, we make it trustworthy as long as all nodes, including at least one honest node, attest it. Hence, a corrupted node has no choice but to collaborate with honest nodes and to conform to the different processes, as otherwise the execution would eventually be aborted. To enforce our requirement, we make use of the following techniques:

1. **Collaborative validation:** a set of nodes, containing at least one honest node, validates via *cryptographic signatures* an information or a fact called f. The nodes must be provided with, or already possess, sufficient knowledge to be able to check f. If all the signatures are *consistent* —i.e., sign the same f—, then f is deemed trustworthy. If at least one signature is not consistent, then f must be discarded and the execution aborted.
2. **Collective knowledge:** every node stores some information on all the other nodes such that CI impersonation can be prevented. Similarly, by storing the query budget of all nodes, we prevent an attacker from abusing her query budget. This requires a collaborative validation.
3. **Collaborative, verifiable process:** a set of nodes, containing at least one honest node, can execute a given algorithm, built in such a way that its output can be validated by all the participants (with sufficient knowledge). For instance, the verifiable random number generation protocol (see Appendix A) allows generating collaboratively a verifiable random value if at least one honest

node participates. In the following, we leverage this verifiable random number protocol as a basis for the collaborative verifiable selection of the query actors. This process is sketched in Section 6.5 and detailed in [40].

6.2 The Need for Efficient, Localized Decisions

These techniques meet Requirement 3, preventing the attacks considered but raise major efficiency/scalability issues.

Assuming an attacker controls up to C corrupted nodes, then both the *Collaborative validation* and the *Collaborative verifiable process* would require at least $C+1$ nodes to ensure that one of them is honest. For instance, with an attacker controlling up to 1% of 10^6 nodes, each node verifying a fact has to perform $2 \times (10^4 + 1)$ asymmetric crypto-operations to check the signature and the certificate of each signatory!

Similarly, for *Collective knowledge*, nodes must perform a full broadcast of any modification of a query budget and maintain a full mesh overlay [40], which is extremely costly in practice and would render irrelevant the DHT overlay.

Finally, asking $C+1$ nodes to attest, for instance, the query to prevent an attacker from altering it is, from a privacy standpoint, counter-productive: an honest querier is almost guaranteed to broadcast it to attackers thus hindering the protections we set up in the previous section.

To maintain an efficient and scalable system, it is thus essential to drastically reduce the number of nodes involved in all these security processes. This can be done through localized decision processes by leveraging again the *imposed location* in the DHT, slightly modifying our requirement.

Indeed, thanks to the imposed location, leading to a uniform distribution of nodes in a DHT, 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 compute then the probability of having **at least k corrupted colluding nodes** and tailor both k and the region size to make the probability lower than a *security threshold*. By setting the threshold sufficiently low, we can consider that having at least k corrupted colluding nodes "never" occurs and thus limit the number of nodes involved in each process to k ($\ll C$). We thus reformulate Requirement 3 to incorporate this probabilistic approach:

> **Requirement 3': Collaborative probabilistic proof.** Any "information" that leads (directly or indirectly) to the transmission of sensitive data must be (i) attested and (if applicable) generated through a collaborative process involving, **with a very high probability**, at least one honest node, and (ii) the participation of at least, one honest node in this process must guarantee its correctness — i.e., it has the power to invalidate it.

Since the localized decisions are taken by the nodes situated inside a specific DHT region, we generalize this notion and define a *legitimate node* as follow: Given a region R in the virtual space of a DHT, for any node n_i we say that n_i is *legitimate* w.r.t. R if and only if it is located within R, i.e., $\texttt{hash}(k_{pub_{n_i}}) \in R$. Legitimate nodes can store information, verify and sign facts, or participate in collaborative processes. We modify the proposed techniques as follows:

1. **Collaborative *local* validation:** instead of needing $C+1$ consistent signatures for a fact f, the signatories are selected in a reduced DHT region and thus minimized to k legitimate nodes.
2. ***Local* knowledge:** instead of broadcasting the query budget to the entire network, the "neighbors" of the Querier are responsible for storing, checking and attesting it. The same occurs with the node locations: each node stores the local topology of the network nodes "around" it.
3. ***Localized* collaborative, verifiable process:** instead of needing $C+1$ participants to generate a verifiable random number or a verifiable list of actors, we select them in a reduced DHT region allowing to reduce their number to k legitimate participants.

6.3 Insertion in the DHT

The main issue regarding the insertion of concepts comes from corrupted nodes impersonating CIs. To prevent it, applying the *Collaborative local validation*, the Node Proxy now requires the CI neighborhood to attest CI's legitimacy. The CI thus asks k legitimate nodes called *CI's Legitimate nodes* (CILs) and located in a small region centered on the concept's location, to verify and attest the fact that the CI is actually the node responsible for the concept (see Fig. 11).

Our fact f, here is that, by construction of the DHT, the CI must be the closest node to the concept. The CI can choose any k CILs to validate f. CILs have the necessary knowledge since they store the local topology of the network. In addition, k is tailored to guarantee the presence of an honest node H among the CILs. Thus, if f is false, i.e., a node N is closest than CI, then H knows N and H will not sign f, thus aborting the process. NP must check (i) that the k signatures are consistent, (ii) that the k signatories' certifi-

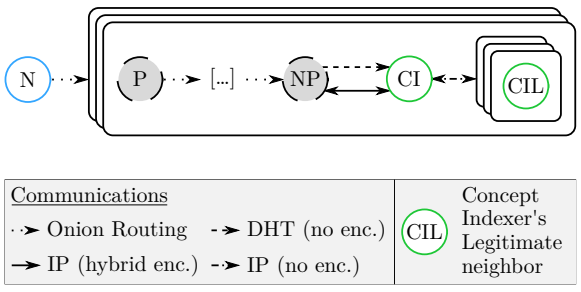

Fig. 11: Insertion of a concept in the DHT

cates are valid and prove that they are legitimate and obviously (iii) that f designates CI (i.e., f has not been altered).

6.4 New Roles for Query Processing (w.r.t. Req. 3')

In the same spirit, the nodes around the Querier are called *Querier's Legitimate nodes* (QLs) and allow to (i) verify and attest the validity of a query w.r.t the Querier query budget, and (ii) bootstrap the random selection of actors by participating in the generation of a verifiable random number.

The first task prevents a corrupted node from abusing the system and freezes the query without disclosing it, effectively blocking *query manipulation* attacks. Additional verifications of the query could be done but are out of the scope of this paper. As in the previous threat models, the random number is used to designate the Actor Selector. In coordination with the AS, the *Actor Selector's Legitimate nodes* (ASLs) generate the list of actors, sign it and, through their signatures, link it with the query.

The next section explains in more details how we compute the probabilistic guarantees and how it can be leveraged to both select the actors and prevent query manipulation.

6.5 Detailed Protocol

Probabilistic guarantees. Having an imposed and uniform distribution of nodes throughout the network — which applies indistinctly to honest and corrupted nodes — we can estimate the number of nodes in a region:

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 — uniformly distributed in the virtual space. The probability, $\mathbb{P}_L$, of having at least m legitimate nodes in R is [40]:

$$\mathbb{P}_L(m_{\geq}, N, rs) = \sum_{i=m}^{N} \binom{N}{i} \times rs^i \times (1-rs)^{N-i} \tag{7}$$

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 $\binom{N}{i}$ combinations of generating this node partitioning. The probability of having in R at least m nodes 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 equation 7. □

The same reasoning applies to obtain the probability, $\mathbb{P}_C$ of having at least k corrupted colluding nodes in R; $C(<N)$ being the maximum number of corrupted colluding nodes:

$$\mathbb{P}_C(k_{\geq}, C, rs) = \sum_{i=k}^{C} \binom{C}{i} \times rs^i \times (1-rs)^{C-i} \tag{8}$$

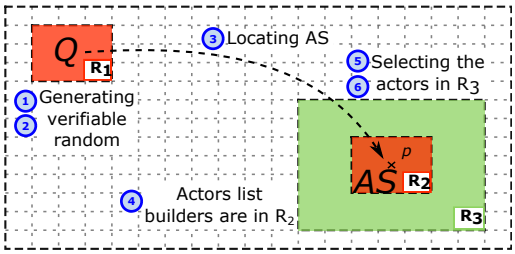

Fig. 12: Verifiable selection of actors protocol

We can notice that this probability does not depend on the region center because of the uniform distribution and is thus valid for any region of size rs.

Furthermore, by correctly choosing the values for k and rs we can make $\mathbb{P}_C$ lower than a given *security threshold* α. In other words, by ensuring that $\mathbb{P}_C$ is lower than α, we "guarantee" that at least one node is honest among the k.

Generating the list of actors. As mentioned earlier, the best achievable protection is obtained when actors are randomly selected and the selection cannot be influenced by attackers, i.e., the average number of corrupted selected actors in the ideal case is $A_{ideal_C} = A \times C/N$. Thus, the impact of a collusion attack remains proportional with the number of colluding nodes, which is the best situation given our context. The protocol proposed below (see Fig. 12) achieves, on average, this ideal case and requires $2k$ signature verifications to check its validity (versus $2 \times (C+1)$) with $k \ll C$.

Protocol 5. Verifiable selection of A actors protocol

1. Q selects k QLs in a region R_1 of size rs centered on Q — such that we have probabilistic guarantees to "always" have at least one honest QL.
2. The k QLs, collaboratively generate RND_Q (verifiable random number generation protocol — see Appendix A).
3. Q hashes RND_Q to obtain a location p in the DHT virtual space and contacts, through the DHT, the node managing this location: the *Actor Selector* (AS).
4. The AS, in turn, selects k ASLs in a region R_2 of size rs centered on p — such that we have probabilistic guarantees to "always" have at least one honest ASL.
5. The k ASLs collaboratively generate another verifiable random number, RND_{AS} and a *Candidate List* (CL) of actors: each ASL_i provides a local candidate list (CL_i) from its cache of nodes ($Cache_i$). Each candidate must belong to a region R_3 centered on p, whose size rs_3 is such that R_3 includes at least A nodes with very high probability.
6. Coordinated by the AS, each ASL_i checks RND_Q, computes the union of all CL_i to obtain CL, sorts CL and keeps the A first actors (sorting is done on $k_{pub_j} \oplus RND_{AS}$ where k_{pub_j} is the public key of node $j \in CL$). They finally sign the list of actors which concludes this protocol.

Steps 1 to 3 relocate, randomly, the selection of actors. Steps 4 and 5 prevent Q from manipulating the relocation. Finally, step 5 and 6 takes care of generating the list of actors. The details of this protocol can be found in [40,38].

The presence of an honest node among the selected QLs and ASLs is the root of security: the honest QL ensures that the relocation is random while the one in ASL ensures that the relocation process was not tampered with, that enough genuine candidates are provided in step 5 and lastly that corrupted ASLs cannot ignore some genuine actor candidates.

Finally, as stated, $2 \times k$ signature verifications are required to verify this list of actors: the certificates of the k ASLs that signed the list, and the k signatures of the list.

Preventing query manipulation. The core idea is to associate the query and the list of actors: to each query corresponds a unique list of actors, thus preventing two attacks: reusing a favorable list of actors and generating a large quantity of list of actors to obtain a favorable one. If a list is tied to a query then, by definition, it cannot be reused. Plus, as each node has a limited query budget, an attacker would exhaust her budget before obtaining a favorable outcome.

However, associating the query to the list of actors must be done carefully: in accordance with the *compartmentalization* technique (see Section 5.1), we should not give actors and intermediary nodes more knowledge than what they need. For instance, although a hash allows detecting any alteration, it requires the raw data to be checked and is, as is, inappropriate. Fortunately, a *Merkle Hash Tree* (MHT) (see Appendix A) solves this limitation since its does not require to send additional meaningful information to any node but simply a correct MHT representation of the query.

Besides altering the query or manipulating the list of actors, an attacker could also retain some information to isolate a Target. For example, a TF could transmit the Local Query to all but one Target, wait a certain period of time, and eventually forward it to the isolated Target hoping that her result would find its way back to the Querier (revealing the identity of the Target). Adding a *timestamp* to the query and defining a validity time frame after which a query should be discarded is an effective way of preventing this issue.

Summarizing, first, the query budget is checked by the k QLs. Then, to prevent any manipulation, the query is associated to the list of actors via k signatures that attest: (i) the root hash of the MHT representation of the query, (ii) the list of actors and (iii) a timestamp.

The detailed protocol is illustrated on Figure 13 and described next highlighting only the additions with regards to the passive attack protocol (i.e., mainly checks, signatures and their verifications).

Protocol 6. DISPERS^{HRC}: Active attack compliant protocol

1. Q asks k legitimate neighbors (QLs) to validate its query and generate RND_Q to locate an AS.
2. Q request the AS to generate a list of actors, providing the signatures of the QLs. If everything is valid, the ASLs and the AS, generate, sign and send the list of actors to Q (see Protocol 5).
3. Q sends to all the actors their query parameters with the corresponding signatures by the ASLs. *All the actors verify the signatures that attest the parameters validity.*
4. Q looks up, in the DHT, each CI managing a share of each concept of the *tp*.
5. The CIs check the signatures of the query and list of actors before contacting the PSs and TFs.
6. The PSs proceed as in the passive attack protocol: reconstruct the $\overline{TIP}$s, apply $\overline{tp}$, sample the Targets, send to the TFs the set of k_{tmp} of the sampled Targets.
7. The TFs proceed as in the passive attack protocol: reconstruct the (TIP, k_{sym}) and transfer to the Ts (via the BP) the lq and the list of DAs with the adequate signatures.

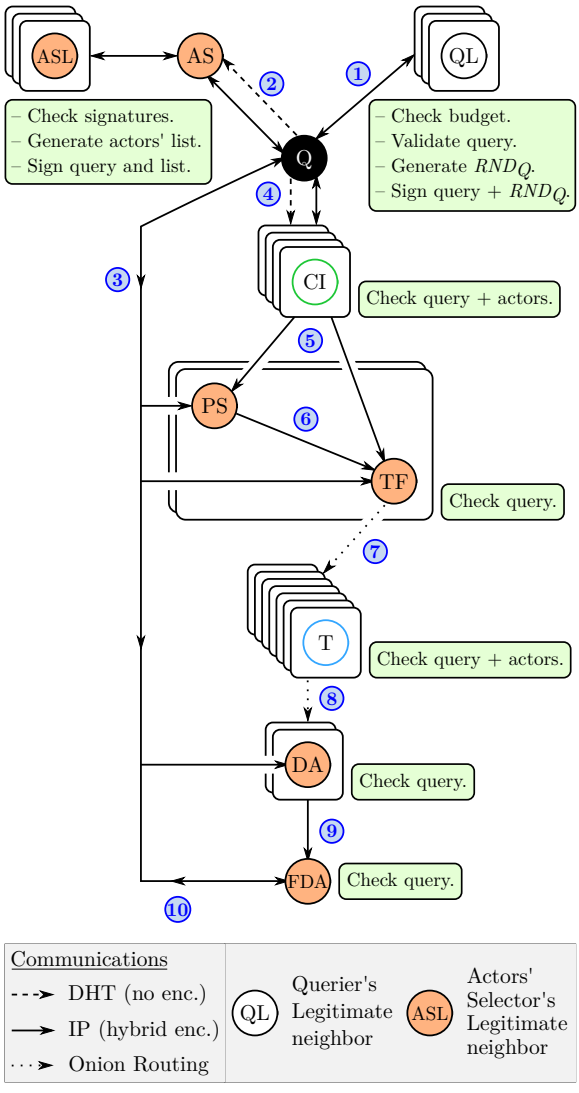

Fig. 13: DISPERS^{HRC} protocol overview

8. The Targets check the validity of the query and of the list of DAs. If everything is valid they then send their local results to the DA (via the AP).
9. The DAs check Q certificate, apply $\overline{aq}$ on the local results, and send the partial results to the FDA.
10. The FDA checks Q certificate, applies $\overline{aq}$ to compute the final result and sends it to Q.

We now discuss a few important considerations regarding the actor selection protocol.

Sparse DHT regions: Despite the uniform distribution of nodes on the DHT virtual space, there could be sparse DHT regions. This can have a negative impact during the selection of k QLs in R_1 (or k ASLs 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 cost of the protocol. On the other hand, setting rs too small can lead to situations in which nodes have less than k legitimate neighbors in their R region and as such cannot participate in the actor selection protocol (as Querier or Actor Selector). For this reason, we provide a table of couples (k_i, rs_i), named *k-table* which gives several increasing values of k with increasing region sizes, computed thanks to $\mathbb{P}_L$ and $\mathbb{P}_C$ (equations 7 and 8). It allows any node to find k_i legitimate neighbors in the region of associated rs_i size keeping the probability of having k_i or more colluding nodes below α (security threshold). Thus, the k-table optimizes the protocol cost and warrants that any node can act as Q or AS.

Choosing R_3 region size: Choosing a too small rs_3 has a negative impact on the system performance. If the ASLs cannot find enough nodes in R_3, they can attest it (e.g., in step 5 of protocol 5) and the AS 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) the 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 is not an option since the maintenance cost of the cache increases proportionally when nodes join or leave the network (see Section 7.5).

Joining the network and $Cache_i$ validity: 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 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 (a recurrence proof can be established).

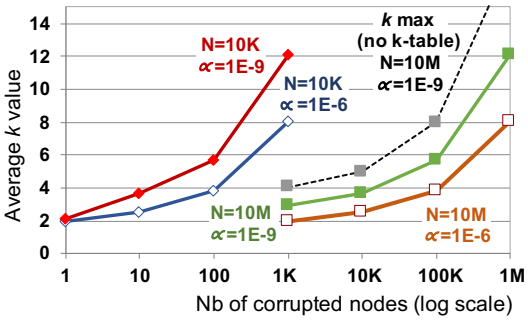

Fig. 14: k versus C (N and α vary)

Failures and disconnections: In the cases of unexpected failure of a QL, ASL or AS, either RND_Q or the list of actors 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 (see also Section 7.5). If a node, selected as actor fails, the impact is mainly on the result quality since part of the results is missing. To maintain a good result quality, the sampling size could be increased in accordance with an estimate of the failure ratio.

6.6 Security Analysis

Protocol 6 can offer the exact same level of protection as in the passive attack model (see Section 5.5) iff all the required attestations are valid, i.e., there is at least one honest node among the k selected legitimate nodes. Therefore, we study in this section the variations on the k value with a large spectrum of possible system configurations.

We have run simulations to understand the variation of k with regards to the total number of nodes, N (10^4 or 10^7), the maximum number of colluding nodes, C (up to $N/10$), and the security parameter, α (10^{-6} or 10^{-9}). We included the value of $N/10$ colluding nodes to understand its impact, even if it is not realistic: it would lead to large disclosure even with an optimal random actor selection protocol.

To obtain these results, we first computed for each C and N the associated k-table. Then, for each configuration and for each node, we computed the minimal value for k with respect to the k-table and α, and we finally averaged the values. We also plotted the value of k without the k-table (gray curve) to highlight the benefit it brings.

Fig. 14 sums up our findings and offers many insights. First, the actors' selection protocol is very scalable w.r.t. N: the values for k are identical for small (10^4) and large net-

work (10^7 nodes), independently of α if we consider the *percentage* of colluding nodes and not the absolute values. Indeed, scaling N and C in the same proportion leads to reduce rs accordingly. Second, k increases slowly when $C < N/100$: k remains smaller than 6 even with $\alpha = 10^{-9}$. Third, α has a small influence on k: decreasing it by three orders of magnitude increases k by only 1 unit. Lastly, the k-table optimization is important: it allows reducing k by 1 unit up to 6 units (for 10% colluding nodes, not shown on the graph).

6.7 Conclusion

Our most advanced protocol combines the protections of the Passive attack and Tamper-proof protocols with *collaborative probabilistic proofs* in order to obtain a scalable and trustworthy execution in the presence of corrupted nodes. Indeed, by checking as little as k signatures, nodes are able to check the validity of the list of actors and query parameters before transferring any sensitive data, thus reaching the lower-bound on leakage with near-certainty as with DISPERS$^{\text{H}}$.

7 Experimental Results

We first define the platform and used metrics in Section 7.1, then describe in Section 7.2 the experimental parameters and how security parameters can be automatically configured. We analyze the experimental results varying several parameters in Sections 7.3, 7.4 and 7.5. Finally, we summarize these results and discuss the setup costs in Section 7.6.

7.1 Evaluated Protocols, Platform and Metrics

Our goal is to evaluate the — quite complex — DISPERS system, composed of a very large number of PDMS nodes executing the proposed protocols over a Chord DHT [66]. Hence, our experimental evaluation is focused on the efficiency[4] and the scalability of DISPERS$^{\text{H}}$, DISPERS$^{\text{HR}}$ and DISPERS$^{\text{HRC}}$ considering respectively the Tamper-proof, Passive attack and Active attack threat models, and varying the parameters impacting security and/or performance. We cannot quantitatively compare our propositions to other strategies given the lack of similar systems (see Section 8).

To evaluate DISPERS, we follow the same general approach as in the related works [66,57,42,59,68,67,75,28], i.e., our results are based on a simulator which creates a logical DHT between simulated nodes. Indeed, simulators were used to evaluate the performance of the state-of-the-art structured DHTs (such as Chord [66] and CAN [57]) and systems that leverage P2P DHTs, such as in the distributed information retrieval area [59,68,67,75,28], to take a few examples closer to our context. The main difficulty to experiment with a P2P network is actually related to its potential very large scale. Simulation thus makes possible the evaluation of systems with millions of nodes and many varying parameters. While a small scale implementation could be interesting (to have, e.g., some real measurements), it is out of reach since it is not compatible with P2P techniques, would lead to abnormal settings (e.g., few nodes storing a large number of concept indexes, too low security thresholds, too few actors) and probabilistic guarantees would fail.

With respect to performance metrics, when evaluating distributed protocols, two aspects should be considered: (i) at the network level, the number of hops (i.e., the path length) or the number of exchanged messages are preferred to time metrics since both offer a more objective view of performance for a large scale distributed system wherein nodes exhibit heterogeneous connection speed and bandwidth used [66,57,68,67,75,28]. In some cases (e.g., when significant amounts of data are transmitted between nodes), the required bandwidth (or bytes per query) is also measured by the simulators [59,75]; and (ii) at the node level, the node CPU resources must be considered both in terms of individual and total resource consumption. For instance, the evaluation of the well known Tor protocol [21] accounts the *asymmetric crypto-operations* which are, by far, the most expensive operations. Our simulator follows the same approach as in the above mentioned related works by capturing two main metrics: (i) at the network level, we consider the *number of exchanged messages* as the most important metric (compared to, e.g., the message time latency or the message size); (ii) at the node level, to measure the impact of security on the PDMS CPU resources, the simulator counts the asymmetric crypto-operations. Hence, we do not consider in the CPU cost the other operations performed by the nodes (e.g., extraction of the *TIPs* lists by CIs, target list computation by PSs/TFs, local query execution by Ts, or aggregates computation by DAs/MDA), which are generic computations having a very small impact on performance — much cheaper and less frequent operations in our protocols compared to the asymmetric crypto-operations —. For each metric, we compute the *ideal latency*, considering that everything that can be done in parallel is actually done in parallel (e.g., messages exchanged in the network). To ease the analysis, we consider that PDMSs cannot process crypto-operations in parallel (which is the case in single core devices). We also compute the *total work* which gives an idea of the global system load incurred by a query. Finally, our simulator outputs the load per node during a query, allowing to check if the load is well balanced or not on the network. Overall, these metrics are more pertinent than absolute time values

[4] The interest reader can refer to the DISPERS demonstration [39] and the associated video (see `https://tinyurl.com/dispers-hrc`) for more qualitative aspects, and to [38] for a practical implementation in CozyCloud [19].

Param.	Description	Values	Default
N	number of nodes	10K to 10M	1M
C	max nb. of coll. nodes	1 to N/10	N/100
Co	number of concepts	1 to 10	3
$\|T\|$	nb. of sampled targets	500 to 2000	1000
α	static security threshold	10^{-6} or 10^{-9}	10^{-9}
β	dynamic security thresh.	10^{-4} or 10^{-6}	10^{-6}
δ	*TIPs* leakage tolerance	10^{-1} or 10^{-2}	10^{-2}
#*AP*, #*BP*	nb of proxies	1 to 20	auto
SD	Shamir degree	5 to 20	auto
k	security degree	2 to 12	auto
A_X	nb. of actors with role X	1 to 512	32
$\|Cache_j\|$	cache size of node j	8 to 32K	48

Table 5: DISPERS Parameters

that are highly dependent on the context (underlying network topology, PDMS node heterogeneity, node bandwidth, network congestion, etc.).

7.2 Parameters and Configuration Tool

The DISPERS parameters are detailed in Table 5 and are divided in four classes discussed below.

System setup (N, C). We consider medium to very large P2P networks, up to 10 million nodes. Note that C represents the maximum number of colluding nodes *controlled by a single attacker* — i.e., the *total number* of corrupted nodes can be much larger than C. Typically, having 10^4 colluding nodes controlled by an attacker is already a highly corrupted system. The maximum value, $C = N/10$, is equivalent to state-size attack and only included to highlight the behavior of DISPERS in extreme conditions.

Query definition (Co, $|T|$). Given our performance metrics, we can abstract queries using only two parameters which may impact the system security and performance. First, the number of concepts Co of the query Target Profile *tp* impacts the number of CIs that need to be contacted, i.e., one for each secret share of each concept. Our Co parameter covers thus a wide spectrum of queries, from very basic ones (1 concept, e.g., `profession`|*researcher*) to very complex (with up to 10 concepts with logical connectors). Second, the number of sampled targets T has a strong impact on query cost. We consider queries selecting from 500 to 2000 targets allowing for statistically significant results. Before the query execution, our simulator selects randomly the Querier node, the CI nodes and the Target nodes since their location does not impact the protocols. The other query actors are also selected randomly as mentioned in each protocol.

System security/thresholds (α, β, δ, #*AP*, #*BP*, *SD*, k). Fixing the system security parameters is a complex task since it depends on the security analysis. We thus implemented a configuration tool based on the equations of Sections 4.5, 5.5 and 6.6, to derive their values based on three security thresholds. Therefore, the system security configuration becomes basic and only requires users to indicate the maximum collusion attack level (C) and the desired thresholds (α, β, δ, or predefined settings) to guarantee the expected security.

1. α was introduced in Section 6.2 and is used to derive k and *SD*, i.e., an attacker should "never" be able to (i) find k colluding nodes in a DHT region; and (ii) control nodes storing the same concept such that he obtains t shares (on a total of *SD* shares). We fixed $t = SD - 3$ in our simulation, such that the system can still be used even if 3 CIs are not available (failure or disconnection). The value of 3 offers a good trade-off between robustness and cost, especially for PDMS with good connectivity.
2. β is introduced as a security threshold for full association disclosure (see Section 5.5). It represents the maximum probability of disclosing one or more full associations and is thus used to derive #*AP*.
3. δ is related to the computation of #*BP*. As we have seen in Section 5.5, leaking some *TIPs* is unavoidable when TF nodes are leaking or corrupted. In average, TF nodes leak $|T| \times C/N$. Fig. 9 shows that adding two BPs reduces by a factor of 3 the expected *TIP* leakage. Further addition of BPs reduce marginally this leakage. The number of BPs is computed such that the *TIP* leakage expectation is less than $|T| \times C/N \times (1+\delta)$. For instance, if $\delta = 1\%$, we tolerate 1% more leakage.

Figures 15 and 16 present the output of the configuration tool, i.e., the values of k, *SD*, #*AP* and #*BP* for increasing values of C with a network of 10^6 nodes and two settings: *reasonable* — $\alpha = 10^{-6}$, $\beta = 10^{-4}$ and $\delta = 10^{-1}$ — and *paranoid* — $\alpha = 10^{-9}$, $\beta = 10^{-6}$ and $\delta = 10^{-2}$. We note that (i) the paranoid setting increases all values by a maximum of 2 units with 10^4 colluding nodes, showing a good scalability (w.r.t. the exigence of the setting); (ii) the values for #*AP*, #*BP*, *SD* and k are reasonable up to 10^4 colluding nodes (1% of the network); (iii) k curves have no steps. This is because k is averaged using the k-tables (see Section 6.5). *We use the paranoid setting in the following measurements.*

System tuning (A_W, A_{PS}, A_{TF}, A_{DA}, $|Cache_j|$). The number of actors (A_X) must be carefully tuned since it impacts the parallelism degree during the execution. Studying its variation is the topic of Section 7.4, while the size of the cache also has a great impact which is studied in Section 7.5.

7.3 Varying the Number of Colluding Nodes

Figures 17, 18, 19, and 20 present the latency and total work for both metrics. We note that: (i) all steps are due to the corresponding ones in the configuration tool output (#*AP*, #*BP*, *SD*, k); (ii) the communication latency (Fig. 17) is almost the same for DISPERS$^{\text{H}}$ and DISPERS$^{\text{HR}}$ because DHT lookups

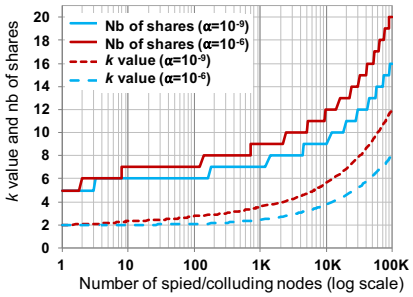

Fig. 15: Config. output: k and SD

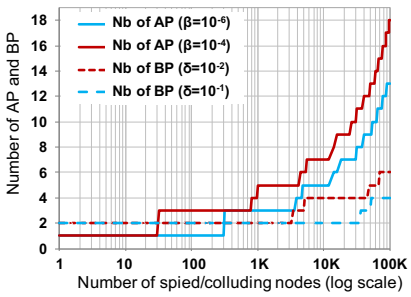

Fig. 16: Config. output: #AP, #BP

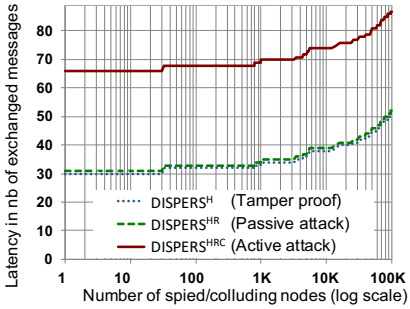

Fig. 17: Communication latency vs C

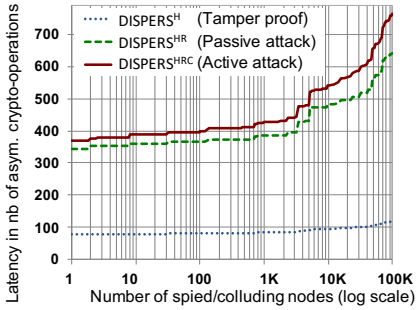

Fig. 18: Crypto latency vs C

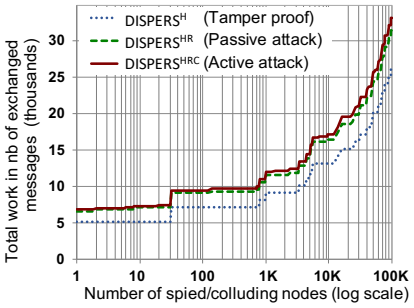

Fig. 19: Communication total work vs C

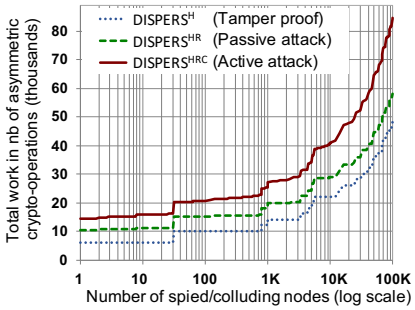

Fig. 20: Crypto total work vs C

(finding AS and CIs) are done in parallel, while it is much higher for DISPERS^{HRC}. Indeed, with DISPERS^{HRC}, finding the CIs can only be done once the actors are selected, thus in sequence. In addition, DISPERS^{HRC} includes more steps to compute collaboratively the randoms and the list of actors. (iii) For the crypto-latency (Fig. 18), there is a gap between DISPERS^{H} and DISPERS^{HR}, mainly due to the CIs which store shares. There is also a smaller gap between DISPERS^{HR} and DISPERS^{HRC}, increasing with the number of colluding nodes. This gap is the consequence of the different checks ($2 \times k$ operations) that depend on k, which in turn depends on C. (iv) The total work graphs (Figures 19 and 20) show bigger steps which are correlated to the increased number of proxies. Indeed, each time we add one BP or AP, the number of crypto-operations and exchanged message are increased by 1 for *each sampled target*, thus largely impacting the total work. Some optimization could be provided here (e.g., using a single proxy for many targets), but they may allow the attacker to distinguish proxies from targets, thus reducing their usefulness. We will address this issue in future works.

Fig. 21 presents the impact of each security technique (and thus each requirement) on the cryptographic total work for DISPERS^{H}, DISPERS^{HR} and DISPERS^{HRC}, with an increasing number of colluding nodes ($10^2, 10^3, 10^4$). As expected, we observe that the proxies overhead is important, each incurring $|T|$ asymmetric encryptions/decryptions. Also, the overhead of BPs is smaller than the one of APs. Indeed, 2 to 3 BPs are required to hide the Targets' *TIPs* (w.r.t. β) while the number of APs varies from 3 to 7 to hide the full associations (w.r.t. δ). Requirement 2 induces reasonable overhead related to the increased number of CIs in the execution plan (due to the shares) and the number of secured channels between CIs and PSs/TFs (each CI must communicate with all PSs and TFs). Finally, the impact of Requirement 3' is important but largely reduced thanks to the probabilistic nature of the proofs and the localized decisions. For instance, with 10^4 colluding nodes, using non-probabilistic proofs would have incurred more than 20M cryptographic operations ($2 \times |T| \times (C+1)$, just for the targets!), compared to around 13K with probabilistic proofs. Note that the total cost for DISPERS^{H} is almost exclusively related to proxies: there are few secured channels (between Q, AS, CIs and Ws) and, without proxies, the communications between the Ws and Ts are symmetrically encrypted.

Fig. 21 must be correlated with Fig. 22 which shows the distribution of the total cost (comm. and crypto.) on the different operators for $C = 10K$. The histogram shows cumulative costs per operator type (the number of instance of each operator type is indicated below the X axis). Clearly, the majority of the cryptographic costs (in red) is concentrated "around" the Targets: T performs $2 \times k$ checks and the onion routing to the DA, while the TF, BP and AP do onion routing between TFs and Ts. However, this work is evenly distributed on the proxies (4000 BPs and 7000 APs with 1000 Ts) and on several TFs (32), guaranteeing a proper distribution of the cryptographic load on the PDMSs. We can obtain a similar conclusion with the communication costs (in blue) with two notable differences: each T only sends 1 message

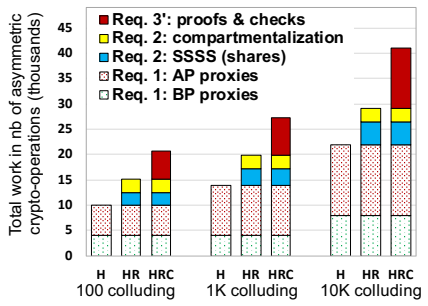

Fig. 21: Crypto cost distribution

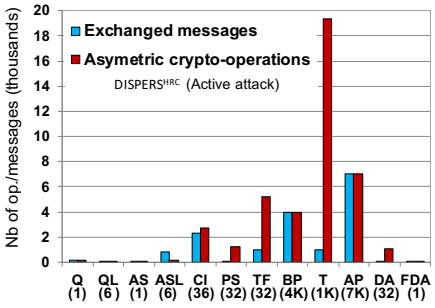

Fig. 22: Cumulative costs/actor

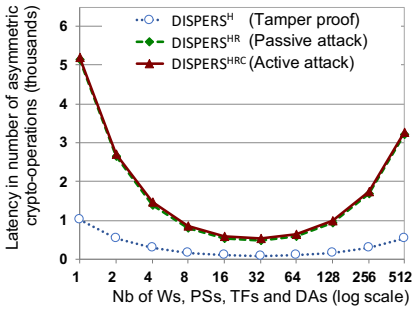

Fig. 23: Crypto latency vs A

(its local result) to the first AP, and TFs only send one message to the first BP for each target.

7.4 Varying the Number of Actors

Figures 23, 24 and 25 present respectively the cryptographic latency, the communication total work and the cryptographic total work for DISPERS^{H}, DISPERS^{HR} and DISPERS^{HRC}. The communication latency is not shown since the number of actors does not impact the ideal latency (communications are done in parallel). Fig. 26 shows the detail of the cryptographic cost per operator for the DISPERS^{HRC} protocol to better understand its behavior. Note that Fig. 26 does not show cumulative costs as Fig. 22. The figure for DISPERS^{HR} (not shown) is similar with less actors (no QL, no ASL) and smaller cost for CI and T (no verification).

We can see in Fig. 23 that the cryptographic latency reaches a minimum around 32 actors (i.e., 32 Ws, 32 PSs, 32 TFs and 32 DAs) independently of the protocol.

Let us focus first on DISPERS^{H}: increasing the number of workers increases the load of Q, as it has to establish the secure channels with all Ws, but decreases the load of each W, as they communicate with less Targets. The global impact on latency is however reduced since DISPERS^{H} uses basic proxies (a single encryption per Target). The impact on the total work (communication or cryptographic costs) is also reduced since most of the cost is concentrated on the BP, T and AP, since they are multiplied by $|T|$.

For DISPERS^{HR} and DISPERS^{HRC}, Fig. 26 shows that increasing the number of PSs, TFs and DAs increases the cost of Q (secure channels), of the CIs (each CI communicates with all PSs and TFs) and of the FDA (which communicates with all DAs), but decreases the cost of the TFs (which contacts less Targets, thus making much less encryption for the onion routing) and DAs (which receive less results and thus create less secure channels). With very few actors, the TFs become the bottleneck as they must contact many Targets. With a large number of actors, the CIs are the bottleneck as they send their lists of *TIPs* to all the PSs and TFs.

Interestingly, the total cost of DISPERS^{HR} and DISPERS^{HRC} (Figures 24 and 25) sshows notable increase when there are more than 32 actors, due to the communications between the CIs, PSs and TFs. For instance, with 512 PSs and TFs, we have around 36K secure channels between CIs, PSs and TFs explaining the large gap between DISPERS^{HR} and DISPERS^{H} with many actors. For the total communication cost, the gap between DISPERS^{HRC} and DISPERS^{HR} increases with the number of actors. This is due to the collaborative selection of actors which is expensive with a large number of actors and is realized by k ASLs in parallel, versus a single AS for DISPERS^{HR}. Fig. 26 also shows that with 32 actors, each actor performs less than 180 asymmetric crypto-operations, which is a reasonable and well distributed load.

7.5 Varying Other Parameters

Having a simulator allowed us to vary independently each parameter of Table 3 and to observe its impact on the metrics. The most interesting results were presented above, but we summarize below the impact of the other parameters.

Varying the size of the network (N) has a marginal impact on performance thanks to the DHT and to the small number of lookups made during a query (mainly to find the CIs). Increasing N and C in the same proportion basically changes nothing, while when we increase N while keeping C constant, the system becomes more efficient because the values of k, $\#BP$ and $\#AP$ decrease as these values depend on the ratio of spied/leaking nodes, i.e., $c = \frac{C}{N}$.

Varying the node cache size ($Cache_j$): The cache is part of the *local knowledge* of a node. Obviously, maintaining this information up-to-date has a cost which must be minimized while avoiding the risk of triggering a *relocation* of the actor selection process (see Section 6.5). Previous simulation results presented in [40] have shown that choosing a cache size smaller or even equal to the required number of actors (A) triggers many query relocations (e.g., almost one relocation, in average when $Cache_j = A$, and almost 14 relocations when it's 3/4 of it!). Nevertheless, our measurements showed that having $Cache_j > 2 \times A$ reduces the relocation ratio to almost 0, making its impact insignificant.

Varying the failure ratio: The detailed measurements were presented in [40]. The goal was to evaluate the impact of the

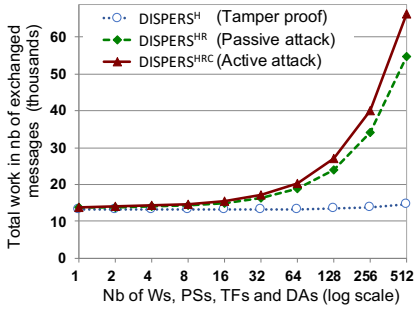

Fig. 24: Communication total work vs A

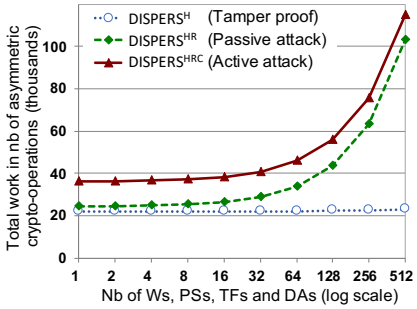

Fig. 25: Crypto total work vs A

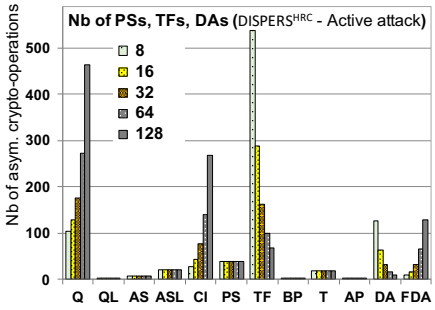

Fig. 26: Crypto cost/actor vs A

cache size in the presence of node disconnections and, more generally, the impact of disconnections. To observe it, we simulated disconnections and measured the global impact on maintenance costs when nodes disconnect (and reconnect) every x hours. Our results showed that (i) considering very large caches (e.g., 10^4) is too costly and consumes a large portion of the computing power of the entire system just to maintain it up to date; (ii) we can safely set the node cache size around $2 \times A$. It almost never triggers relocations and leads to a reasonable maintenance cost (less than 1 crypto-operation/node/minute for $x = 24$ hours, see [40]).

7.6 Conclusion of Experimental Results

The main conclusions of the experimental results are:

- The increasing complexity of the protocols (DISPERS^{H}, DISPERS^{HR}, DISPERS^{HRC}) is reflected by their increasing costs as shown in Fig. 21. However, this increase is quite reasonable given the different optimizations described in Sections 5.4 and 6.5.
- A simple configuration tool allows fixing $\#AP$, $\#BP$, SD and k based on the equations provided in the security analysis sections and on meaningful security thresholds α, β, δ. The outputs of the configuration tool show a good "scalability" w.r.t. the requirements of the setting, i.e., the thresholds. The values for $\#AP$, $\#BP$, SD and k are reasonably small for a highly corrupted network (e.g., 10^4 colluding nodes for 10^6 total nodes).
- DISPERS scales well w.r.t. C (the number of colluding nodes), and has good performance (good latency and well distributed total work). Additionally, as a consequence of the previous remark, the impact of the increase of C stays reasonable even with 10^4 colluding nodes.
- The tuning of the number of PSs, TFs, and DAs is relatively easy to do. We perform several experiments to find the minimal latency that we observe in Fig. 23 and found experimentally that the corresponding value is proportional to the square root of the number of Targets.
- Increasing in the same proportion N and C has a marginal impact, while increasing N keeping C constant makes the system more efficient (since c is reduced, $\#AP$, $\#BP$ and k also are).
- The node cache size should be around twice the number of actors (A) to avoid query relocation. In addition, such a tuned size for the cache leads to reasonable maintenance costs in case of node failure.

Setup costs for a real deployment of DISPERS. One important advantage of DISPERS is that it is built on top of a DHT without requiring any adaptation. Hence, it can be deployed as an app on top of any existing structured DHT (e.g., Chord [66], CAN [57] or Kademlia [42]). Also, DHTs are proven technology (offering many benefits such as full decentralization, scalability, fault tolerance, load balancing –see also Appendix B) and have been quite successful even in practice for specific use cases (e.g., file-sharing applications like BitTorrent, Gnutella or GNUnet). All this plays in favor of DISPERS requiring low and predictable overhead in terms of implementation and setup cost.

On the other hand, DISPERS requires interfacing with the PDMS devices mainly at two levels: for getting the user profile and for getting the local results for system queries. In turn, this calls for the standardization of the PDMS interface, e.g., similar to a relational SQL database. It also requires interactions with the PDMS owner [14] that has to set the appropriate sharing rules and exposed user profile. We argue that all these issues, essential for a deployment, are not specific or more complex for DISPERS than for a centralized system, e.g., like Prio [18] or SMCQL [9] which need to collect users data either offline [9] or at query time [18].

Finally, once the data sharing policies are defined, the users themselves do not need to be online (i.e., live interaction) unless they want to query the system. Since the PDMS device plays the role of a personal server (e.g., under the form of a plug computer), it should be mostly connected allowing its owner to run personal or distributed data-oriented apps at all times. Hence, its connectivity should not be an issue. Obviously, in the context of DISPERS, the more PDMSs are connected and the more stable is their connection, the better it is for finding pertinent data for the distributed queries.

8 Related Works

This section describes the related works at different *levels*: the DHT, the distributed indexes, the security tools, the active attack countermeasures, and finally the overall approach.

DHT security. Several works focus on DHT security [74] considering the following attacks: (i) Sybil attack: an attacker generates numerous false (and malicious) DHT nodes to disturb the protocols. Sybil attacks can be addressed using node certification thanks to a PKI [15]. (ii) Eclipse attack: an attacker attempts to control most of the neighbors of honest nodes to isolate them. The best countermeasure [74] is to constrain the DHT node identifiers. Again, using a central authority to provide verifiable identifiers is the simplest yet most effective way of achieving this goal [66]. (iii) Routing and storage attacks: a malicious node in the path of a lookup request can disrupt the DHT routing, claiming to be the recipient, answering fake data or erroneously forwarding the request, thus denying the existence of a valid key. The mechanisms employed to negate these attacks are based on redundancy at the storage and routing levels [74]. DISPERS leverages the idea of imposed node location based on a trustworthy PKI certificate like in the aforementioned works. Routing and storage attacks are not directly addressed but the verifications added to the active attack protocol use collaborative probabilistic proofs to ensure a correct execution.

Distributed indexes. Several works consider indexing documents, profiles or even databases on top of a DHT using a distributed version of inverted indexes [67] as for our *concept index*. Enhancements were proposed to reduce the number of lookup operations [31], to minimize the index size [64] or to index compact database summary [28]. These proposals do not consider security issues and are closely related with the type of indexation (e.g., for keyword searches), needing further work for security and profile indexing.

Security tools. DISPERS relies on classical security tools (see Appendix A), sometimes slightly adjusted to achieve better system efficiency or to better distribute the system load (e.g., adapted onion routing). Also, message anonymization has been extensively used in secure distributed data aggregation protocols (e.g., [53,54]). However, many works consider a trusted third party, i.e., the anonymizer. To relieve our system from this (strong) assumption, we employ message forwarding through proxy node chains whenever anonymization is required, which is similar to onion routing in distributed systems [58]. Finally, SSSS is frequently used in distributed systems to protect data-at-rest [12,27]. It provides a fully-secure solution to our distributed profile indexing problem with a better trade-off between efficiency and redundancy than alternative secrete sharing schemes [56].

Active attack countermeasures. In the active attack model, corrupted nodes exhibit the so-called *Byzantine* behavior [36]. Several distributed protocols leverage the Byzantine fault-tolerant *consensus protocols* such as distributed file systems [16] or permissioned blockchain systems [41]. Such protocols ensure a correct execution if at least $2/3$ of the participants are honest. However, they also involve a high network overhead since they require communication between all nodes (i.e., $O(N^2)$ number of messages), which makes them unsuitable to large-scale distributed systems. In DISPERS, we employ a different approach based on a CSAR-like protocol [8], i.e., collaborative proof, in which a single honest node is sufficient to guarantee the correctness of a proof. However, directly applied to our fully-distributed setting, the collaborative proof is not scalable with the number of colluding nodes leading to very large cryptographic and communication overhead. Hence, we propose a probabilistic approach reducing drastically the protocol cost which can then be applied even with a huge number of colluding nodes.

Secure distributed data aggregation. Secure data aggregation in distributed environments has been a hot research topic for many years, leading to the following approaches. (i) Secure MPC protocols based on homomorphic encryption [54,12], secret sharing [27] or randomization [10]. Such solutions offer strong (formal) security guarantees but generally do not scale to large number of nodes, lack genericity w.r.t. the computation function [61] and cannot handle node targeting. (ii) Local differential privacy (LDP) has gain significant momentum in recent years due to its major advantage compared with classical DP, i.e., it does not require a trusted third party. Existing works address problems such as machine learning [77], marginal statistics [76] or basic statistics based on range queries [17]. However, LDP accentuates the tension between utility and privacy protection since it generally requires more noise to achieve the same level of protection as with classical DP [3]. Hence, this can either affect utility or require a very large number of participants to reduce the impact of noise (which is the opposite of our node targeting approach). (iii) To overcome some of the limitations of MPC or DP, several works propose using secure hardware at the user-side to address, e.g., SQL aggregation [69], spatio-temporal aggregation [53], or privacy-preserving data publishing [2]. This approach is generic w.r.t. the computation function but the existing solutions do not address the node selection problem and generally consider a tamper-proof attack model or a very small number of corrupted nodes. Compared with the above mentioned classes of solutions, DISPERS targets (very) large-scale (e.g., nationwide or beyond) and fully-distributed systems by leveraging the state-of-the-art DHT communication overlay. The system security is built on two complementary principles: guaranteed random actor selection (to reduce the probability of a leakage) and task compartmentalization (to reduce the impact of a leakage). These principles open the way for distributed query processing with minimized leakage and reasonable and scalable security overhead.

9 Limitations and Other Challenges

This section discusses the limitations of this work in more details.

Computation integrity. DISPERS does not consider the issues related to the correctness of the contributed data or, in the Active Attack threat model, of the computation process. Indeed, in the Tamper-proof and Passive attack threat models, PDMSs follow the protocols and cannot alter the computations input/output: they have the means to attest that a given computation was correctly performed [33,34]. With fully corrupted PDMSs, this problem requires a complementary in-depth study. While existing solutions, such as Prio [18], could be used to verify that nodes' contributions are within some predefined interval, it will not warrant correct results. Indeed, the computation is fully distributed and data processors may also be corrupted. Note that, in our context, not all aspects of the execution can be modified. The leeway an attacker has is limited to the addition of bogus results or to the inclusion of more nodes in the query execution. This study, mainly due to its sheer size, falls outside of the scope of this paper.

Data aggregation and secure MPC. As argued in Section 1, the PDMS context naturally leads to a fully distributed architecture since the data is hold at the user side, stored in PDMSs. This is different from the centralized or federated architectures typically used in MPC in which a handful of powerful servers hold large collections of user data (e.g., SMCQL [9], Conclave [73] and Obscure [27]) or collect user data at query time (e.g., Prio [18]) and then apply costly (although optimized) aggregation algorithms based on garbled circuits (e.g., [9]) or secret sharing (e.g., [27]). In DISPERS, data aggregation is performed by randomly selected nodes which cannot reasonably apply the evoked MPC approaches given the limited resources of user devices. Furthermore, the architectural difference also implies a different threat model. MPC solutions typically consider an honest-but-curious attacker (e.g., SMCQL or Conclave) or an honest querier (e.g., Obscure). In DISPERS both the data aggregators and the querier can be corrupted and, even worse, colluding. However, for classical aggregate functions, more secure distributed aggregation protocols, adapted to the DISPERS architecture can be envisioned. This is the focus of our future work extending DISPERS (preliminary results in [45]).

Improving the DHT overlay network latency. In practice, in heterogeneous P2P systems, the latency of a message exchange between two nodes of a DHT can greatly vary. Moreover, the number of physical routing hops between two nodes can also impact the network latency. Some DHT overlays (such as CAN [57]) propose to optimize DHT communication latency by bringing the DHT logical overlay "closer" to the physical network. Such optimization is inapplicable in our system for security reasons: the node location in our DHT is imposed and cannot be influenced. However, even if these improvements are out of reach for security and node heterogeneity reasons, in practice, reducing the number of messages remains the main factor for the system's scalability.

Inference attacks on the final result. In DISPERS, we follow the typical approach of the works in secure data aggregation area (see e.g., Prio [18] or [12]), which focus on protecting the aggregation process itself, except for what attackers can infer from the statistical results computed by the system and any additional knowledge they may have. Such inference attacks are indeed possible [71], but defending against them is outside the scope of this paper. We note though that some basic defense mechanisms of DISPERS (e.g., requiring having a minimum number of targets for a query, random sampling of the targets when their number is higher than the set threshold, the query budget) can help mitigating attacks on the query results, but need to be complemented to enforce the system security in this regard.

10 Conclusion

Personal Data Management Systems arrive at a rapid pace allowing users to share their personal data within large P2P communities. While the benefits are unquestionable, the important risks of personal data leakage and misuse represent a major obstacle on the way of the massive adoption of such systems. This paper is one of the first efforts to deal with this challenging issue. To this end, we proposed DISPERS, a fully-distributed P2P system laying the foundation for secure, efficient and scalable execution of distributed computations. By considering a palette of realistic threat models, we analyzed the fundamental security and efficiency requirements of such a distributed system leading to three security requirements that must be fulfilled to minimize the private data disclosure: (i) hidden communications, (ii) random dispersion of data, and (iii) collaborative proofs.

Although some leakage is unavoidable with passive or active attacks, we showed that our approach makes it possible: (i) to have an integrated solution covering both node targeting and data aggregation, thus going beyond basic data aggregation by considering the important problem of the pertinence of the contributor nodes to a query (having potentially a major impact on both the quality of the result and the query cost); (ii) to consider the full range of attacks, i.e., from only communication spying to fully-corrupted nodes; (iii) to have an efficient and scalable system with an adjustable trade-off between the security level and its cost.

This work opens the way for several interesting research problems. In particular, we focus currently on investigating a richer data and query model that can be applied to distributed machine learning algorithms, and complementary strategies to improve the data protection especially during

the aggregation phase. Also, understanding the duality that exists between the computation integrity and the data confidentiality in the presence of malicious nodes is another important and challenging problem that we plan on studying.

Acknowledgment. This research was partially supported by Cozy Cloud, France, by the ANR PersoCloud grant ANR–16–CE39–0014 and by the PEPR iPoP.

References

1. T. Allard, N. Anciaux, L. Bouganim, Y. Guo, et al. Secure Personal Data Servers: a Vision Paper. *PVLDB*, 3(1-2), 2010.
2. T. Allard, B. Nguyen, and P. Pucheral. MET$_A$P: revisiting Privacy-Preserving Data Publishing using secure devices. *Distributed and Parallel Databases*, 32(2), 2014.
3. M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and A. Pazii. Local Differential Privacy on Metric Spaces: Optimizing the Trade-Off with Utility. In *IEEE CSF*, 2018.
4. N. Anciaux, P. Bonnet, L. Bouganim, B. Nguyen, et al. Personal Data Management Systems: The security and functionality standpoint. *Information Systems*, 80, 2018.
5. N. Anciaux, L. Bouganim, P. Pucheral, Y. Guo, et al. MILo-DB: a personal, secure and portable database machine. *Distributed and Parallel Databases*, 32(1), 2014.
6. N. Anciaux, L. Bouganim, P. Pucheral, I. S. Popa, et al. Personal Database Security and Trusted Execution Environments: A Tutorial at the Crossroads. *PVLDB*, 12(12), 2019.
7. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2), 2010.
8. M. Backes, P. Druschel, A. Haeberlen, and D. Unruh. CSAR: A Practical and Provable Technique to Make Randomized Systems Accountable. In *NDSS*, 2009.
9. J. Bater, G. Elliott, C. Eggen, S. Goel, et al. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB*, 10(6), 2017.
10. A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi. Personalized and Private Peer-to-Peer Machine Learning. In *AISTATS*, 2018.
11. S. L. Blond, P. Manils, C. Abdelberi, M. A. Kâafar, et al. One bad apple spoils the bunch: Exploiting P2P applications to trace and profile tor users. In *USENIX LEET*, 2011.
12. K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM CCS*, 2017.
13. R. Carpentier, I. S. Popa, and N. Anciaux. Reducing data leakage on personal data management systems. In *IEEE EuroS&P*, 2021.
14. R. Carpentier, F. Thiant, I. Sandu Popa, N. Anciaux, et al. An Extensive and Secure Personal Data Management System using SGX. In *EDBT*, 2022.
15. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, et al. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, 36(SI), 2002.
16. M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *OSDI*, 1999.
17. G. Cormode, T. Kulkarni, and D. Srivastava. Answering Range Queries Under Local Differential Privacy. *PVLDB*, 12(10), 2019.
18. H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, 2017.
19. Cozy Cloud. A smart personal cloud to gather all your data. (see `https://cozy.io/en`), 2021.
20. Y.-A. De Montjoye, E. Shmueli, S. S. Wang, and A. S. Pentland. OpenPDS: Protecting the privacy of metadata through safeanswers. *PloS one*, 9(7), 2014.
21. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX SSYM*, 2004.
22. J. Douceur. The Sybil attack. In *Int. Workshop on Peer-to-Peer Systems*, 2002.
23. European Commission. Proposal for a regulation on european data governance (data governance act), com/2020/767. [eur-lex], 25 October 2020.
24. European Parliament. General Data Protection Regulation. (see `https://gdpr-info.eu/`), 2018.
25. P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, et al. Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys Tutorials*, 17(2), 2015.
26. M. Gulati, M. J. Smith, and S.-Y. Yu. Security enclave processor for a system on a chip, 2014. US Patent 8,832,465.
27. P. Gupta, Y. Li, S. Mehrotra, N. Panwar, et al. Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. volume 12, 2019.
28. R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Summary management in P2P systems. In *EDBT*, 2008.
29. G. Heiser and K. Elphinstone. L4 Microkernels: The Lessons from 20 Years of Research and Deployment. *ACM Trans. Comput. Syst.*, 34(1), 2016.
30. W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301), 1963.
31. Y. Joung, L. Yang, and C. Fang. Keyword search in DHT-based peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 25(1), 2007.
32. A. Kermarrec and F. Taïani. Want to scale in centralized systems? Think P2P. *J. Internet Services and Applications*, 6(1), 2015.
33. R. Ladjel, N. Anciaux, P. Pucheral, and G. Scerri. A Manifest-Based Framework for Organizing the Management of Personal Data at the Edge of the Network. In *ISD*, 2019.
34. R. Ladjel, N. Anciaux, P. Pucheral, and G. Scerri. Trustworthy Distributed Computations on Personal Data Using Trusted Execution Environments. In *TrustCom*, 2019.
35. S. Lallali, N. Anciaux, I. S. Popa, and P. Pucheral. Supporting secure keyword search in the personal cloud. *Information Systems*, 72, 2017.
36. L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3), 1982.
37. S. Lee, E. L. Wong, D. Goel, M. Dahlin, et al. πbox: A platform for privacy-preserving apps. In *NSDI*, 2013.
38. J. Loudet. *Distributed and Privacy-Preserving Personal Queries on Personal Clouds*. PhD thesis, Versailles University, 2019.
39. J. Loudet, I. S. Popa, and L. Bouganim. DISPERS: Securing Highly Distributed Queries on Personal Data Management Systems. *PVLDB*, 12(12), 2019.
40. J. Loudet, I. S. Popa, and L. Bouganim. SEP2P: Secure and Efficient P2P Personal Data Processing. In *EDBT*, 2019.
41. S. Maiyya, V. Zakhary, M. J. Amiri, D. Agrawal, et al. Database and Distributed Computing Foundations of Blockchains. In *SIGMOD*, 2019.
42. P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Int. Workshop on Peer-to-Peer Systems*, 2002.
43. A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. 1996.
44. R. C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO*, volume 293, 1987.
45. J. Mirval, L. Bouganim, and I. S. Popa. Practical fully-decentralized secure aggregation for personal data management systems. In *SSDBM*, 2021.
46. MyData Global. Empowering individuals by improving their right to self-determination regarding their personal data. (see `https://mydata.org`), 2020.

47. M. Nanni, G. L. Andrienko, A. Barabási, C. Boldrini, et al. Give more data, awareness and control to individual citizens, and they will help COVID-19 containment. *Trans. Data Priv.*, 13(1), 2020.
48. Nextcloud. The self-hosted productivity platform that keeps you in contro. (see `https://nextcloud.com`), 2021.
49. A. Nilsson, P. N. Bideh, and J. Brorsson. A survey of published attacks on intel SGX. *CoRR*, abs/2006.13598, 2020.
50. R. Nithyanand, O. Starov, P. Gill, A. Zair, et al. Measuring and mitigating as-level adversaries against tor. In *NDSS*, 2016.
51. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, 4th Edition*. Springer, 2020.
52. S. Pinto and N. Santos. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.*, 51(6), 2019.
53. I. S. Popa, D. H. T. That, K. Zeitouni, and C. Borcea. Mobile participatory sensing with strong privacy guarantees using secure probes. *GeoInformatica*, 25(3), 2021.
54. R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li. Privacy and accountability for location-based aggregate statistics. In *CCS*, 2011.
55. C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A Secure Database Using SGX. In *IEEE S&P*, 2018.
56. M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *J. ACM*, 36(2), 1989.
57. S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, et al. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.
58. M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4), 1998.
59. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, 2003.
60. M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted Execution Environment: What It is, and What It is Not. In *Trust-Com/BigDataSE/ISPA (1)*, 2015.
61. E. Saleh, A. Alsa'deh, A. Kayed, and C. Meinel. Processing over encrypted data: between theory and practice. *ACM SIGMOD Record*, 45(3), 2016.
62. Secure Data Hub. Output Confidentiality Rules. (see `https://www.casd.eu/wp/wp-content/uploads/Output_Confidentiality_Rules.pdf`), 2021.
63. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11), 1979.
64. G. Skobeltsyn, T. Luu, I. P. Zarko, M. Rajman, et al. Web text retrieval with a P2P query-driven index. In *SIGIR*, 2007.
65. Solid. All of your data, under your control. (see `https://solidproject.org/`), 2021.
66. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, et al. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM*, 31(4), 2001.
67. C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *NSDI*, 2004.
68. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *ACM SIGCOMM*, 2003.
69. Q. To, B. Nguyen, and P. Pucheral. Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture. *ACM Trans. Database Syst.*, 41(3), 2016.
70. J. C. Tomàs, B. Amann, N. Travers, and D. Vodislav. RoSeS: a continuous query processor for large-scale RSS filtering and aggregation. In *ACM CIKM*, 2011.
71. J. Unnikrishnan and F. M. Naini. De-anonymizing private data by matching statistics. In *IEEE Allerton*, 2013.
72. G. Urdaneta, G. Pierre, and M. V. Steen. A survey of DHT security techniques. *ACM Computing Surveys (CSUR)*, 43(2), 2011.
73. N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, et al. Conclave: Secure multi-party computation on big data. In *EuroSys*, 2019.
74. Q. Wang and N. Borisov. Octopus: A Secure and Anonymous DHT Lookup. In *ICDCS*, 2012.
75. Y. Yang, R. Dunlap, M. Rexroad, and B. F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. In *INFOCOM*, 2006.
76. Z. Zhang, T. Wang, N. Li, S. He, et al. CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy. In *ACM CCS*, 2018.
77. K. Zheng, W. Mou, and L. Wang. Collect at Once, Use Effectively: Making Non-interactive Locally Private Learning Possible. In *ICML*, volume 70, 2017.
78. ';–Have i been pwned. Check if you have an account that has been compromised. (see `https://haveibeenpwned.com/`), 2021.

A Background on Cryptography

Symmetric encryption is computationally efficient but requires a symmetric encryption key k_{sym} known beforehand by both parties. On the contrary, **asymmetric encryption** is a demanding operation that relies on a pair of keys: the *private* key, k_{priv}, and its matching *public* key, k_{pub}. To avoid man-in-the-middle attacks, k_{pub} must be certified. **Hybrid encryption** uses asymmetric encryption to securely exchange a symmetric encryption key and combines the advantages of both encryption schemes. To ensure *forward secrecy* [43], a new symmetric key is used for each communication session. The widely used TLS protocol is based on hybrid encryption and provides also integrity, and authenticity of the communicating parties.

A **cryptographic hash function** [43], referred as $\texttt{hash}()$, is a one-way function that maps a data of arbitrary size to a fixed size bit string (e.g., 256 bits), is resistant to collision and provides a uniform distribution of its outputs.

A **digital signature** [43] can be used to prove that a data d was produced by an entity E (*authentication*) and has not been altered (*integrity*). A signature contains the encryption of $\texttt{hash}(d)$ using k_{priv_E} and the certificate of k_{pub_E}, $cert_E$. Anyone can check a signature by checking the certificate, decrypting the encrypted hash, and finally comparing the result with $\texttt{hash}(d)$ (recomputed by the verifier).

Shamir's Secret Sharing Scheme (SSSS) [63] consists in dividing some data d into n shares $d_1, \ldots, d_n$ in such a way that: (i) knowledge of any t ($t \leq n$) or more shares makes d easily computable; but (ii) knowledge of any $t-1$ or fewer shares leaves d protected (not even providing any information about it). t is called the *threshold* value (see Section 8) and is set to resist to $n-t$ shareholders failures. The low, polynomial complexity of SSSS (i.e., Lagrange interpolation) for both secret decomposition and reconstruction, makes it an ideal solution for a fully-distributed system like DISPERS in which any PDMS node has to securely store its profile in the DHT or can be selected as actor node (Profile Sampler or Target Finder) to recompose a secret. Note that DISPERS employs the basic SSSS and does not require more advanced (and much costlier) operations such as string-matching on secret-shares or order-preserving secret-sharing, e.g., as used in [27].

Anonymous communications can be obtained by using **onion routing** technique [58]. The sender selects all the *routers* and asymmetrically encrypts the message "in layers", as an onion. Each router decrypts one layer and discovers dynamically the next router up to the destination.

A **Merkle Hash Tree** (MHT) [44] is a tree data structure for which leaf labels are hashes of data blocks $d_1, \ldots, d_n$, and the remaining tree nodes are labeled with the hash of their children's labels. The root of the tree is digitally signed allowing to check the integrity of any of the data blocks, computing the intermediary hashes, starting from the leaf, going up to the root and verifying that the computed root matches the signed one. MHTs are particularly useful to check the integrity of a given block d_i without disclosing the others data blocks, but only the intermediate hashes in the MHT.

A **verifiable random number generation protocol** is a protocol which allows n nodes to produce a random value R, while guaranteeing that none of the n nodes can choose or influence the value of R. This is made possible if, at least, one of the n nodes is honest. A version of this protocol is described in details in our previous work [40] and is adapted from [8] which includes a formal proof. It roughly unfolds as following: (i) each node selects a random value r_i and commits on it by sending $\texttt{hash}(r_i)$ to a coordinator; (ii) the list of hash values, L, is disclosed by the coordinator to the n nodes; (iii) each node then checks that $\texttt{hash}(r_i) \in L$ and, if so, sends r_i and a signature of L back to the coordinator. R is finally obtained by computing a XOR of the n individual random values. An attacker controlling $n-1$ nodes cannot influence R since these nodes cannot change their r_i, committed with $\texttt{hash}(r_i)$. Thus, the random value of a single honest node is enough to obtain a truly random final value.

B Background on Distributed Hash Tables

A **Distributed Hash Table** (DHT) in a P2P network [51] offers an optimized solution to the problem of locating the node storing a specific data item. The DHT offers a basic interface allowing nodes to store data, i.e., $\texttt{store}(\texttt{key}, \texttt{value})$, or to search for certain data, i.e., $\texttt{lookup}(\texttt{key}) \rightarrow \texttt{value}$. DHT proposals share the concepts of keyspace or DHT virtual space (e.g., a 256 bits string obtained by hashing the key or the node ID with the SHA256 algorithm), space partitioning (mapping space partitions to nodes, using generally a distance function), and overlay network (routing tables and strategies allowing reaching a node, given its ID). For instance, the virtual space is represented as a multi-dimensional space in CAN [57], as a ring in Chord [66] or as a binary tree in Kademlia [42] and is uniformly divided among the nodes in the network. Thus, each node is responsible for the storage of all the $(\texttt{key}, \texttt{value})$ pairs where the key falls in the subspace it manages. The `store` and `lookup` operations are fully distributed: DHTs do not require any central coordination. They are scalable, fault tolerant and provide a uniform distribution of the data.