SEP2P: Secure and Efficient P2P Personal Data Processing

Julien Loudet, Iulian Sandu-Popa, Luc Bouganim

SEP2P: Secure and Efficient P2P Personal Data Processing

Julien Loudet[1,2,3]
[1]Cozy Cloud, France
julien@cozycloud.cc

Iulian Sandu-Popa[3,2]
[2]INRIA Saclay, France
<fname.lname>@inria.fr

Luc Bouganim[2,3]
[3]University of Versailles, France
<fname.lname>@uvsq.fr

ABSTRACT

Personal Data Management Systems are flourishing allowing an individual to integrate all her personal data in a single place and use it for her benefit and for the benefit of the community. This leads to a significant paradigm shift since personal data become massively distributed. In this context, an important issue needed to be addressed is: how can users/applications execute queries and computations over this massively distributed data in a secure and efficient way, relying exclusively on peer-to-peer (P2P) interactions? In this paper, we motivate and study the feasibility of such a pure P2P personal data management system and provide efficient and scalable mechanisms to reduce the data leakage to its minimum with covert adversaries. In particular, we show that data processing tasks can be assigned to nodes in a verifiable random way, which cannot be influenced by malicious colluding nodes. Then, we propose a generic solution which largely minimizes the verification cost. Our experimental evaluation shows that the proposed protocols lead to minimal private information leakage, while the cost of the security mechanisms remains very low even with a large number of colluding corrupted nodes. Finally, we illustrate our generic protocol proposal on three data-oriented use-cases, namely, participatory sensing, targeted data diffusion and more general distributed aggregative queries.

1 INTRODUCTION

The time of individualized management and control over one's personal data is upon us. Thanks to smart disclosure initiatives (e.g., BlueButton [9] and GreenButton in US, MesInfos [16] in France, Midata [25] in UK) and new regulations (e.g., the Europe's new General Data Protection Regulation [27]), 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 (e.g., Personal Data Servers [1], Personal Information Management Systems, Personal Data Stores [14], Personal Clouds [20]) and industry [12, 26, 33]. Their goal is to offer a data platform allowing users to easily store and manage into a single place data directly generated by user devices (e.g., quantified-self data, smart home data, photos, etc.) and data resulting from user interactions (e.g., user preferences, social interaction data, health, bank, 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 holds the promise of unlocking new innovative usages.

Let us consider three emblematic distributed applications based on large user communities which could greatly benefit from the PDMS paradigm: (1) mobile participatory sensing apps [36], in which mobile users produce sensed geo-localized data (e.g., traffic, air quality, noise, health conditions) to compute spatially aggregated statistics benefiting the whole community; (2) subscription-based or profile-based data diffusion apps [38], in which PDMS users provide preferences or exhibit profiles in order to selectively receive pertinent information; and (3) distributed query processing over the personal data of large sets of individuals [37], in which users contribute with their personal data and issue queries over the globally contributed data (e.g., computing recommendations, participative studies).

However, these exciting perspectives should not eclipse the security issues raised by the PDMS paradigm. Indeed, each PDMS can store potentially the entire digital life of its owner, thereby proportionally increasing the impact of a leakage. Hence, centralizing all users' data into powerful servers is risky since these data servers become highly desirable targets for attackers: huge amounts of personal data belonging to millions of individuals could be leaked or lost as illustrated by the recent massive attacks (e.g., Facebook, Yahoo or Equifax). Besides, such a centralized solution makes little sense in the PDMS context in which data is naturally distributed at the users' side [19].

Alternatively, recent works [4, 14, 20, 33] propose to let the user data distributed on personal trustworthy platforms under users' control. Such platforms can be built thanks to the combination of (1) a Trusted Execution Environment (TEE) (i.e., secure hardware such as smart cards [1] or secure micro-controllers [4, 5, 20], ARM TrustZone [18], or Intel SGX [29]) and (2) specific software (e.g., minimal Trusted Computing Base and information flow control [22, 29]). In this paper, we follow this approach and consider that a PDMS is a dedicated personal device that the user possesses and is secured thanks to TEE hardware.

In addition, as in many academic and commercial approaches [33], we assume that the PDMS personal device offers a rather good connectivity and availability like, for instance, home-cloud solutions [4, 12, 26, 33] (e.g., a set-top box or a plug computer [4]). Thus, PDMSs can establish peer-to-peer (P2P) connections with other PDMSs, and can be used as data processor in order to provide part of the processing required in distributed applications. Hence, our objective is to study solutions based on a full distribution of PDMSs (called nodes interchangeably) which can act as data sources and data processors and communicate in a peer-to-peer fashion. We discard solutions requiring recentralizing the distributed personal data during its processing, since this would dynamically create a personal data concentration leading to a similar risk as with centralized servers.

Incorporating TEEs considerably increases the protection against malicious PDMS owners. However, since no security measure can be considered as unbreakable, we cannot exclude having some corrupted nodes in the system and, even worse, those corrupted nodes can collude and might very well be undistinguishable from honest nodes, acting as covert adversaries [7]. Also, since data processing relies exclusively on PDMS nodes, and given the very high scale of the distribution which disqualifies secure multi-party computation (MPC) protocols [31], sensitive data leaks are unavoidable in the presence of corrupted nodes,

i.e., some data might be disclosed whenever a corrupted node is selected as a data processor.

The goal of this paper is to assess the feasibility of building a secure and efficient data processing system over a fully distributed network of PDMS housing covert adversaries. To achieve it we provide mechanisms to reduce the data leakage to its minimum, and make the following contributions:

(1) We propose a P2P architecture of PDMSs, called SEP2P (for Secure and Efficient P2P), based on classical Distributed Hash Tables (DHT) and analyze potential data leakages of data sources and data processors. We show that (i) data tasks should be assigned to nodes in a *verifiable random* way, i.e., the assignment cannot be influenced by malicious colluding nodes; and (ii) any data-oriented task, whether it is storage or computation, should be *atomic*, i.e., reduced to a maximum such that it minimizes the quantity of sensitive data accessible by the task.

(2) We focus on the verifiable random assignment problem and propose a generic solution (i.e., independent of the distributed computation tasks) which largely minimizes the verification cost (e.g., 8 asymmetric crypto-operations with a SEP2P network of 1M nodes of which 10K are colluding corrupted nodes).

(3) We experimentally evaluate the quality and efficiency of the proposed protocols. The verifiable random assignment protocol leads to minimal private information leakage, i.e., linear with the number of corrupted nodes, while the cost of the security mechanisms remains very low even with a large number of colluding corrupted nodes.

(4) We address the task atomicity subproblem by providing sketches of solutions for the three classes of applications indicated above. We do not propose full solutions since task atomicity is dependent on the considered class of distributed computation and as such needs to be studied in detail.

Sections 2 to 5 present these four contributions respectively. We finally discuss the related work in Section 6 and conclude the paper in Section 7.

2 SEP2P ARCHITECTURAL DESIGN

2.1 Base System Architecture

SEP2P is a peer-to-peer system and only relies on the PDMS nodes to enable the aforementioned applications. Consequently, each node may play several roles for SEP2P applications:

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

Node role 2. Given the fully-decentralized nature of SEP2P, each node is a potential **data processor**, also called **actor**, providing part of the required processing.

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

2.2 Efficient P2P Data Processing

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

Background 1. A **distributed hash table (DHT)** [23, 30, 34] in a P2P network offers an optimized solution to the problem of locating the node(s) storing a specific data item. The DHT offers a basic interface allowing any node of the network to store data, i.e., $store(key, value)$, or to search for certain data, i.e., $lookup(key) \rightarrow value$. DHTs proposals [23, 30, 34] share the concepts of keyspace or **DHT virtual space** (e.g., a 224 bits string obtained by hashing the key or the node ID), space partitioning (mapping space partitions to nodes, using generally a distance function), and overlay network (set of routing tables and strategies allowing reaching a node, given its node ID). For instance, the virtual space is represented as a multi-dimensional space in CAN [30], as a ring in Chord [34] or as a binary tree in Kademlia [23] and is uniformly divided among the nodes in the network. Thus, each node is responsible for the indexing of all the $(key, value)$ pairs where the key falls in the subspace it manages. Both the data storage and the lookup operations are thus fully distributed in a DHT. DHTs have interesting properties: uniform repartition of the data, scalability, fault tolerance and do not require any central coordination.

Hence, SEP2P leverages the classical DHT techniques as a basis for communication efficiency and scalability.

2.3 Security Considerations

In this paper, we use the terminology of ARM [35] to designate the three attack levels on a PDMS node, i.e., *hack*, *shack* and *lab attacks*. A *hack attack* is a software attack in which the attacker (the PDMS owner or remote attacker) downloads code on the device to control it. A *shack attack* is a low-budget hardware attack, i.e., using basic equipment and knowledge. Finally, a *lab attack* is the most advanced, comprehensive and invasive hardware attack for which the attacker has access to laboratory equipment, can perform reverse engineering of a device and monitor analog signals. Note that shack and lab attacks require a physical access to the device and that TEEs are designed to at least resist hack and shack attacks.

Our threat model considers three security assumptions:

Assumption 1. *Each PDMS is locally secured by using TEE-like technology flourishing nowadays* (e.g., [18, 20, 29]). This assumption is reasonable considering that a PDMS is supposed to store the entire digital life of its owner. A major security feature of TEE technology is to provide isolation, i.e., strong guarantees that the local computation inside the TEE cannot be spied upon, even in the presence of an untrusted computational environment. Hence, to break to confidentiality barrier of a TEE, a lab attack is mandatory. This has an important consequence: *an attacker cannot conduct a successful attack on a remote node, i.e., not under her possession.*

Assumption 2. *Each PDMS device is supplied with a trustworthy certificate attesting that it is a genuine PDMS.* Without this assumption, an attacker can easily emulate nodes in the network, and conduct a Sybil attack [11], mastering a large proportion of nodes (e.g., playing the role of data processor nodes), thus defeating any countermeasure. Note that this does not require an online PKI (the certificate can be attached to the hardware device and not to the device owner).

Assumption 3. *Corrupted nodes by a lab attack behave like covert adversaries*, i.e., they derive from the protocol to obtain private information only if they cannot be detected [7], as detected malicious behavior leads to an exclusion from the system.

2.4 Threat Model

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

It is important to notice that the worst-case attack is represented by the maximum number of colluding nodes in the system (i.e., controlled by a single entity). Corrupting few nodes can lead to some private data disclosure, but this will be very limited in a well-designed system with a large number of nodes. Therefore, an attacker needs to increase the collusion range to fully benefit from the attack (i.e., access a significant amount of private data).

Thereby, the remaining question is: how many colluding nodes could an attacker control in the system? The main difficulty for an attacker is that colluding nodes must remain indistinguishable from honest nodes (see Assumption 3). Since PDMSs are associated to "real" individuals (e.g., by delivering the device only to real users proving their identity), collusions between individuals remains possible (hidden groups) but such collusions cannot scale without being minimally advertised, hence breaking the indistinguishability mentioned above. Thus, wide collusions are extremely difficult to build since it requires significant organization between a very large number of users, which in practice requires an extremely powerful attacker as well as extreme discretion, and are thus the equivalent of a state-size attack. Finally, note that considering a large proportion of colluding nodes (e.g., 10%) is vain as it would inexorably lead to large disclosure whatever the protocol having a reasonable overhead (e.g., outside the MPC scope). Hence, in this paper we consider that a very powerful attacker could control up to a small percentage (e.g., 1%) of the nodes, which corresponds to a wide collusion requiring a lab attack on these nodes as well as a highly organized collusion between the owners of those nodes.

What does the system protect? The objective of SEP2P is to offer the maximum possible confidentiality protection of the user private data under the above considered threat model. Many other issues related to statistical databases (e.g., inferences from results, determining the authorized queries, query replay, fake data injection, etc.) or to network security (e.g., message drop/delay, routing table poisoning [39]) are complementary to this work and fall outside of the scope of this paper. Similarly, the problems related to the attestation and integrity of the code executing distributed computations (e.g., against corrupted nodes that maliciously modify the computation results).

2.5 SEP2P Requirements

Given the considered threat model, we derive in this section the requirements that a SEP2P must address to protect the data privacy of the users. Since we cannot exclude having colluding nodes in the system and since the colluding nodes behave like covert adversaries, private information leakage is unavoidable. Under these conditions, the best countermeasures one can take are: (i) *minimize the risk* of a data leakage, i.e., reduce at most the probability of a leakage to happen; and (ii) *minimize the impact* of a data leakage, i.e., reduce at most the leakage size. Obviously, these countermeasures should not generate overheads that render the system unpractical. This leads to:

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

Requirement (security) 2. Task atomicity. Data tasks should be atomic, i.e., reduced to a maximum such that it minimizes the required sensitive data to execute the task.

Requirement (efficiency) 3. Security overheads. Minimize the number of costly operations, e.g., cryptographic signature verifications or communication overheads, and ensure system scalability with an increasing number of nodes or colluding nodes.

The task atomicity requirement is similar to the principle of compartmentalization in information security, which consists in limiting the information access to the minimum amount allowing an entity to execute a certain task. Typically, a node can execute a subtask without knowing the purpose or the scope of the global task. Dividing a given distributed computation in atomic tasks obviously depends on the precise definition of that computation. Hence, we restrict our analysis in Section 5 to sketches of solutions for the three application classes considered in this paper.

Independently of the distributed protocol chosen to implement some given application, the system must delegate the data-oriented tasks to randomly selected nodes. Therefore, the random selection protocol is generic and constitutes the security basis of any distributed protocol in our system. However, given the considered threat model, it is challenging to design an actor selection protocol that is both secure and efficient. Section 3 addresses this problem while section 4 evaluates the proposed solution.

3 SECURE ACTOR SELECTION

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

Background 2. A **cryptographic hash function** [24] is a one-way function that maps a data of arbitrary size to a fixed size bit string (e.g., 224 bits) and is resistant to collision. An interesting property of hash functions is that output distribution is uniform. In the following, $hash()$ refers to cryptographic hash.

Background 3. A **cryptographic signature** [24] can be used by a node n to prove that a data d was produced by n (authentication) and has not been altered (integrity). The signature is produced by encrypting $hash(d)$ using the private key of n. Any node can verify the signature by decrypting it using the public key of n and comparing the result with $hash(d)$. The signature includes the signer public key certificate, $cert_n$ (see Assumption 2).

We consider a system of N nodes, in which we want to randomly select A actors, despite wide collusion attacks from C corrupted nodes. The main notations are summarized in Table 1.

3.1 Effectiveness, Cost and Optimal Bounds

Ideally, we would want to ensure that all A actors are honest, but this is impossible, since colluding nodes are indistinguishable from honest nodes. Therefore, the best achievable protection is obtained when actors are randomly selected and the selection cannot be influenced by C colluding nodes, i.e., the average number of corrupted selected actors in the ideal case is $A_{ideal_C} = A \times C/N$

N	Total number of nodes in the SEP2P system
A	Number of actor nodes (data processors)
C	Maximum number of colluding nodes ($C \geq 1$)
A_C	Average number of corrupted actors for a given protocol
A_{ideal_C}	Average number of corrupted actors for an ideal protocol
T	Triggering node (starting the execution)
k	Security degree
α	Security threshold
S	Execution Setter node, computing actor list
R_i, rs_i	DHT region R_i of size rs_i

Table 1: Main notations for Sections 3.1 and 3.2

($A_{ideal_C} > 0$). Thus, the impact of a collusion attack remains proportional with the number of colluding nodes, which is the best situation given our context. This guarantees that the attacker cannot obtain more private information than what she can passively get from observing the information randomly reaching its colluding nodes.

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

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

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

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

Note that the security cost considers only the verification of the actor list and not the cost of building the list. The rationale is that the verification cost has a larger impact on the overall performance since the number of verifier nodes can be high in a large distributed system: data sources need to verify the actor list before delivering their data. Other performance related issues (cost of the actor list generation, load balancing, maintenance costs) are discussed in Section 3.6 and 4.

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

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

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

Baseline security-optimal protocol. Proposing an optimal protocol in terms of security is challenging in a decentralized architecture (without any supporting trusted party) and considering covert adversaries. This conjunction leads to a situation where no single node in the system can claim to securely provide a list of actors (the provider itself can be corrupted). The work in [8] proposes the CSAR protocol which provides a secure way to generate a verifiable random value under the condition that there is at least one honest node participating in the distributed protocol. Applying to our context, we can ensure generating a real random value only if there are at least $C + 1$ participating nodes. Also, once we obtain a verifiable random value, we can derive up to A random values by repeatedly hashing the initial value $A - 1$ times. The final step is to map the set of A random values to the nodes. This can be easily done, e.g., by sorting the nodes on their public key and associating the random value to a rank in the sorted list. This protocol has an optimal security effectiveness, i.e., 1, since the actors are guaranteed to be selected randomly. On the other hand, checking the CSAR results requires one signature verification per participant. Thus, the security cost is $C + 1$ asymmetric cryptographic operations per verifier node. Since C can be large, such a solution cannot scale with large systems and wide collusion attackers as it would lead to an extreme verification cost.

Moreover, to achieve these security bounds, both protocols require a full mesh network overlay which is also extremely costly to maintain in practice, especially for large networks. This contradicts the efficiency and scalability requirement formulated in Section 2.5. Using a DHT overlay instead of a full mesh solves the problem of communication efficiency/scalability. However, this will impact the optimal bounds of both protocols. For the first protocol, the security cost increases from 0 to up to A since a verifier node which does not "know" any of the actors has to verify their certificates to be sure that the actors are genuine PDMSs (to avoid Sybil attacks). Similarly, for the second protocol, the security cost increases to $2(C + 1) + A$ for the same reason, i.e., checking that participant and selected actors are genuine PDMSs. Even worse, the optimal security effectiveness can no longer be guaranteed since with a DHT, there is no secure way of associating the random values to the nodes unless using secure DHT techniques [39] with a large impact on performance.

3.2 Overview of the proposed solution

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

Imposed and uniform distribution of node location: the node ID, used when inserting a node in the DHT, is imposed in SEP2P, in a way that leads to a uniformly distributed node location in the DHT virtual space. Consequently, colluding nodes are also evenly distributed in the DHT, thus avoiding spatial clusters. We use extensively this property to drastically reduce the cost of security by taking localized decisions (see below), i.e., limited to the nodes situated in "small" regions in the virtual space. Achieving imposed node location is easy, based on the public

key of the certificate of each node. We compute a cryptographic hash of this key, which is, by construction, uniformly distributed, and use this hash for insertion in the DHT virtual space. The advantages of using the public key are (i) its uniqueness; and (ii) the node location can be checked with a single signature verification.

Probabilistic guarantees: Given the imposed, uniform node location which applies indistinctly to honest and colluding nodes, we can have probabilistic guarantees on the maximum number of colluding nodes in a DHT subspace of a given size, called *DHT region* hereafter. We can compute the probability of having **at least k colluding nodes** (see Section 3.3) and choose the DHT region size such that the probability is very close to 0. In our context, we want to have a probability smaller than α, the security threshold. The main idea is to set α so that the probability of having k colluding nodes in the same region becomes so low that we can consider that it "never happens", e.g., $\alpha = 10^{-6}$ (see Section 4.1). Such a guarantee is used in the protocol sketched below and then detailed in the following subsection.

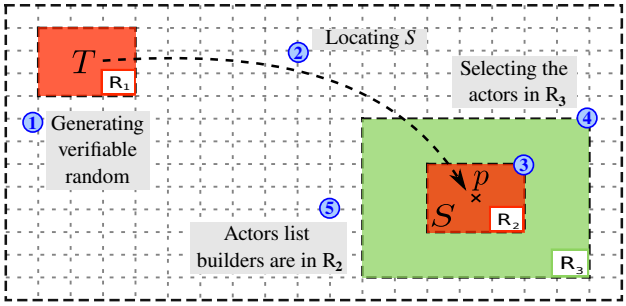

Figure 1: Sketch of verifiable selection

Sketch of verifiable selection protocol of A actors (see Figure 1)

(1) Run a distributed protocol inspired from CSAR [8] to generate a **verifiable random value**, i.e., proven to have been truly randomly generated by k nodes if at least one is honest (see Section 3.4). The k nodes are selected in a DHT region R_1, centered on the triggering node (T), whose region size rs_1 is set such that we have probabilistic guarantees to "never" (probability $< \alpha$) have k or more colluding nodes, i.e., at least one of the k nodes is honest.
(2) Map the hash of that random value into coordinates to define a location p in the DHT virtual space and contact through the DHT the node, called **execution Setter** (S), managing this location.
(3) S then selects k nodes (the actor list builders) in a region R_2, centered on p, using probabilistic guarantees, such that we "never" have k or more colluding nodes. Given the uniform distribution of the node on the virtual space, we have $rs_2 = rs_1$.
(4) Each actor list builder then selects A nodes in a region R_3, centered on p, whose size rs_3 is such that R_3 includes at least A nodes with high probability (see Section 3.6 and Section 4.3 for rs_3 tuning).
(5) Run a **distributed verifiable selection protocol** in the spirit of [8] such that the k nodes selected in (3) can: (i) check the validity of the random value generated in (1); (ii) build the actor list securely; (iii) sign both the random value and the list of A actors. This step is detailed in Section 3.5.

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

3.3 Providing Probabilistic Guarantees

To generate verifiable random values or validate the query actor selection, SEP2P employs distributed computations between a small subset of the nodes thanks to the notion of node legitimacy and probabilistic guarantees defined below using the notations in Table 2.

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

Table 2: Main notations for Sections 3.3 – 3.5

Definition 4. Legitimate nodes. Given a region R in the virtual space of a DHT, for any node i we say that node i is legitimate w.r.t. R iff $hash(kpub_i) \in R$.

To be able to provide probabilistic guarantees as explained in Section 3.2, we need to estimate the number of nodes in a region:

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

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

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

Application to colluding nodes: Let $C < N$ be the maximum number of colluding nodes. We can apply formula 1 to compute the probability, PC of having at least k colluding nodes in R:

$$PC(\geq_k, C, rs) = \sum_{i=k}^{C} \binom{C}{i} \cdot rs^i \cdot (1-rs)^{C-i} \quad (2)$$

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

3.4 Verifiable Random Generation

Our goal is to generate a random value, using k nodes and to guarantee that none of the k nodes can choose the final computed random value (or any of its bits). Any node in the system should be able to check the validity of this random value (i.e., to have proofs that it has been correctly generated). This is possible as soon as at least one of the k nodes is honest, this guarantee being

obtained thanks to equation (2) by choosing the adequate size for the DHT region R and by using k legitimate nodes w.r.t. R.

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

Verifiable random number generation protocol

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

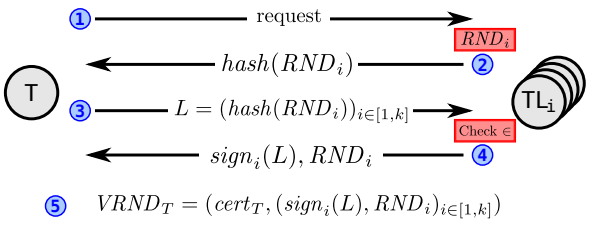

Figure 2: Verifiable random

The above random generation protocol is adapted from [8] which includes a formal proof. Note that the protocol in [8] does not include the notion of node legitimacy and thus needs $C + 1$ participating nodes instead of k. Intuitively, the nodes commit on their selected random value by sending its hash (Step 2), and all the hash values are known by each of the k nodes before providing the final signature (Step 4). Therefore, an attacker controlling $k - 1$ TL_i nodes cannot influence the final random value since these nodes cannot change their random values (committed at Step 2). Thus, the correct random value of a single honest node is enough to obtain a truly random final value RND_T.

To obtain and check the verifiable random value, any node must: (i) check $cert_T$ and compute L by hashing all RND_i; (ii) for $i \in [1, k]$, check $cert_i$, check the legitimacy of TL_i using $cert_T$ and validate $sign_i(L)$. The final random value is $RND_T = RND_1 \oplus RND_2 \oplus \cdots \oplus RND_k$.

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

3.5 Distributed Secure Selection Protocol

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

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

In the following, we assume that each node n in SEP2P keeps a node cache, called $cache_n$, of the IP address and certificate of legitimate nodes w.r.t. a region of size rs_3 centered on node n location. The cache size and the cache maintenance cost are discussed in Section 3.6 and evaluated in Section 4.3.

SEP2P distributed secure actor selection protocol

(1) Generates the verifiable random $VRND_T$ (see Section 3.4).
(2) Maps $hash(RND_T)$ into coordinates and contact S through the DHT.
(3) S contacts any k legitimates nodes w.r.t. R_2, SL_j ($j \in [1, k]$) and sends to each $VRND_T$ (see Section 3.4).
(4) Each SL_j sends $hash(RND_j \parallel CL_j)$ to S, where RND_j is a random number SL_j generates, and CL_j is the set of nodes from $Cache_j$ which are legitimate w.r.t. R_3.
(5) Once S has received the k hashes, it sends back the list L_1 of hashes to all SL_j; $L_1 = (hash(RND_j \parallel CL_j))_{j\in[1,k]}$.
(6) Each SL_j checks that its own $hash(RND_j \parallel CL_j) \in L_1$ and, in the positive case, returns RND_j and CL_j.
(7) S gathers the k messages and sends to all SL_j the list $L_2 = ((RND_j, CL_j)_{j\in[1,k]})$.
(8) Each SL_j does the following:
  (a) Checks $VRND_T$ and computes RND_T (see Section 3.4).
  (b) Checks that each (RND_j, CL_j) from L_2 is consistent with the corresponding $hash(RND_j \parallel CL_j)$ from L_1.
  (c) Computes the union, after removing possible duplicates, of all CL_j to obtain a candidate list of legitimate nodes CL.
  (d) Computes the $RND_S = RND_1 \oplus RND_2 \oplus \cdots \oplus RND_k$.
  (e) Sorts CL on $kpub_n \oplus RND_S$ (where $kpub_n$ is the public key of a node $n \in CL$) and selects the A first candidates to build the actor list AL.
  (f) Checks the legitimacy of AL nodes w.r.t. R_3.
  (g) Signs (RND_T, AL) and sends it to S.
(9) S gathers k results and builds the verifiable actor lists: $VAL = (RND_T, AL, (sign_j(RND_T, AL)))_{j\in[1,k]}$.

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

Steps 3 to 6 are similar to steps 1 to 4 of the verifiable random protocol, except that the signature by SL_j is delayed to Step 8.g. Delaying the signature allows SL_js to check and attest the validity of $VRND_T$ (step 8.a). The protocol cost is increased (since k nodes verify $VRND_T$) but the verifying cost is reduced accordingly since having k SL_js signing RND_T (step 8.g) means that it is correct (remind that at least one of the k SL_js is honest).

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

In **steps 8.f and 8.g**, k SL_js check the validity of the result, i.e., that any actor of AL belongs to R_3 and attest it by signing the results. Note that this check is not necessary for any actor n in AL that was found in k CL_j since this fact attests that at least

one honest node possesses n in its $Cache_j$. Assuming $Cache_j$ contains only genuine nodes (we say that $Cache_j$ is valid - see Section 3.6) and since $rs_3 > rs_2$, most of the actors in AL will be found in k CL_j, thus diminishing drastically the actor list building cost. Actually the validity of $Cache_j$ is necessary to ensure that a colluding node selected as SL cannot hide honest nodes with the hope of having a larger proportion of colluding nodes in AL. Indeed, at least one of the SL is honest and will provide its full $Cache_j$ that will be thus included in CL. We can observe that $Cache_j$ can be actually seen as the relevant part (for node j) of a full mesh network, which offers its benefits without paying the whole maintenance cost.

Let us now concentrate on the work that must be done by the verifier nodes. To check the verifiable actor list (VAL), any verifier node must do: for $j \in [1, k]$, check $cert_j$, check the legitimacy of SL_j using RND_T and validate $sign_j(AL)$. Thus, the verifying cost is limited to k certificate verifications and k signature verifications, i.e., $2k$ asymmetric crypto-operations. We show in Section 4 that k is generally lower than 6.

3.6 Protocol Implementation Details

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

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

Choosing R_1 (or R_2) region size: on the one hand, a small rs leads to a smaller k value, which in turn reduces the protocol verification cost. On the other hand, setting rs too small can lead to situations in which nodes have less than k legitimate nodes in their R region and as such cannot participate in the actor selection protocol (as triggering node or execution setter) which is problematic. For this reason, in SEP2P we provide a table of couples (k_i, rs_i), named **k-table**, which allows any node to find k_i legitimate nodes in the region of associated rs_i size. The k-table is computed thanks to PL and PC (equations (1) and (2)) to ensure that whatever the couple chosen, the probability of having k or more colluding nodes remains equal. The largest k of the k-table corresponds to the region size allowing any node to find those legitimate nodes with a very high probability, i.e., 1-α, while lower values allow to reduce the security cost in denser network regions. Thus, the k-table optimize the overall cost of the SEP2P protocol and warrant that any node can be selected as triggering node or execution setter.

Choosing R_3 region size: Choosing a too small rs_3 has a negative impact on the system performance. If the SLs cannot find enough nodes in R_3, they can attest it (e.g., in Step 8.c in SEP2P protocol) and S can use the k signatures to displace the actor selection to another region (e.g., selected by rehashing the initial RND_T). This mechanism allows the protocol to be executed successfully even if some network regions are sparser. However, there are two drawbacks. First, the cost of the actor selection increases since (part of) SEP2P protocol must be executed twice (or more times). Second, this also introduces an unbalance in the system load since the sparse regions cannot fully take part in data processing. Finally, setting rs_3 to very large values (see Section 4.3) is not an option since the maintenance cost of the cache increases proportionally when nodes join or leave the network.

Joining the network and $Cache_j$ validity: Due to space limitation, we only sketch the joining procedure in the case of a Chord DHT (leaving the network can be easily deduced). As mentioned above, any node must maintain a consistent node cache despite the natural evolution of the network. Thus, a node joining the network must ask its successors and predecessors (Chord DHT) to provide their node cache attested by k legitimate nodes in a region of size rs_1 centered on their location. The new node can then make the union of these caches and keep only legitimate nodes w.r.t R_3 centered on its location. The resulting cache contains only genuine nodes and is thus valid since it has been attested by at least k nodes in a region of size rs_1 centered on the successors or predecessors of the new node (a recurrence proof can be established).

Reusing an actor list: If there is no mechanism that prevents an attacker from reusing an actor list, then she only has to keep generating such lists until she obtains one she deems satisfying. To counter this behavior, we put in place two mechanisms: (i) a timestamp and (ii) a limit to the number of triggered executions a node can make. With (i) we prevent any node from reusing an actor list: TLs and SLs add a timestamp to their signatures which will respectively be checked by the SLs and the data sources. If the timestamp is too distant, the computation is cancelled. Enforcing (ii) is possible thanks to the node cache and the k-table: the TLs solicited by T first check if T chose the smallest possible number of TLs (as their node cache contains, by construction, R_1 centered on T, they are capable of judging), thus forcing T to choose the same TLs. They then only have to monitor and limit the number of queries T does in a given amount of time.

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

4 EXPERIMENTAL EVALUATION

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

4.1 Experimental Setting

Reference methods. To better underline our contributions and to provide a comparison basis, we implemented three strategies in addition to the SEP2P actor selection protocol. We discarded the baseline cost-optimal and security-optimal protocols from the evaluation since the former does not provide any security while the latter is much too costly and not scalable (w.r.t. N and C) to be used in practice. Hence, we used for comparison more advanced actor selection strategies based on these protocols but using our verifiable random generation protocol with k participants (see Section 3.4).

The first two strategies use the verifiable random to designate the execution Setter (S) which freely chooses the actor list (as in the cost-optimal protocol). These strategies differ only in the verification process. The first one, ES.NAV (for Execution Setter, No Actor Verification) requires verifying the legitimacy of S but not of the actors. The second one, ES.AV requires, in addition,

to verify that actors are genuine PDMSs. ES.AV is expected to provide better security effectiveness than ES.NAV at a higher verification cost. The third strategy, M.Hash (for Multiple Hash) is derived from the security optimal protocol, but uses a DHT instead of a full mesh network. Verifiers must check that actors are genuine PDMSs and that they are "near" the random values determined by the initial verifiable random, hashed as many times as there are actors.

Strategy	Description
ES.NAV	Execution Setter with No Actor Verification
ES.AV	Execution Setter with Actor Verification
M.Hash	Multiple Hash (with Actor Verification)
SEP2P	Proposed protocol (Section 3.5)

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

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

Table 3: Strategies, parameters and metrics

Simulation platform. We identified all the parameters that may impact the security and efficiency of the proposed strategies and considered all the metrics (see Table 3) that are worth evaluating to analyze the strengths and weaknesses of the proposed strategies, i.e., security effectiveness and cost, setup cost, scalability, robustness w.r.t. failure or disconnections. Let us note that a real implementation of the SEP2P distributed system is not very useful if we consider the above listed objectives of the evaluation. Also, measuring the scalability for very large systems (e.g., 10M nodes) with many parameters is practically impossible. Therefore, as in most of the works on distributed systems [30, 34], we base our evaluation on a simulator and objective metrics. That is, the latency is measured as the number of asymmetric crypto-operations and exchanged messages between peers instead of absolute time values. This allows for a more precise assessment of the system performance than time latency, which can greatly vary in our context because of the node heterogeneity (e.g., TEE resources or network performance).

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

Security threshold value: Generating several networks and varying the security threshold α, we experimentally observed that for $\alpha = 10^{-4}$, an attacker never controls k or more nodes. However, given the importance of this parameter for the system security, we set $\alpha = 10^{-6}$ and show in Figure 6 the impact of choosing $\alpha = 10^{-10}$ on a small ($10K$) and large ($10M$) network. Indeed, if an attacker could master by chance k colluding nodes in a region of size $rs_1 = rs_2$, then she could completely circumvent the security mechanism of SEP2P since, for example, she can obtain k signatures from these regrouped colluding nodes for an actor list of her willing. Note that increasing α reduces the probability accordingly but increases the verification cost in a logarithmic way (as discussed below in Section 4.3).

4.2 Security Effectiveness vs. efficiency

Figure 3 represents the security effectiveness (Y axis) versus the verification cost (X axis) for the four measured strategies and with $C\%$ varying from 0.001% to 10%. Note that the value of 10% is not realistic: it would lead to large disclosure even with an optimal random actor selection protocol, and as mentioned in Section 2.4, is equivalent to state-size attack. We have however run the simulation with 10% to understand its impact on the security effectiveness and cost.

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

Setup costs: Figures 4 and 5 show the setup costs (Y axis in log scale) in terms of asymmetric crypto-operations and exchanged messages respectively, once more with respect to the verification cost (X axis). Curves with empty symbols represent latency while plain symbols represent total work. The results show that SEP2P is the slowest in latency and has the higher total setup cost for crypto-operations. These "bad" results are the consequence of two design choices: (1) to increase the security effectiveness, we

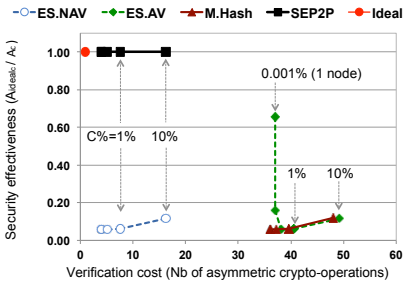

Figure 3: Sec. Effectiveness vs Verification

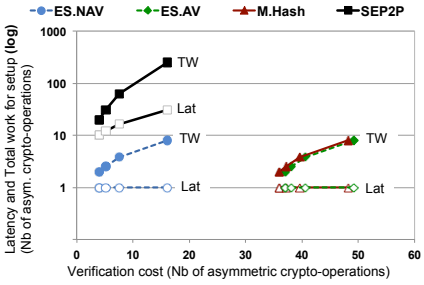

Figure 4: Setup asymmetric crypto-costs

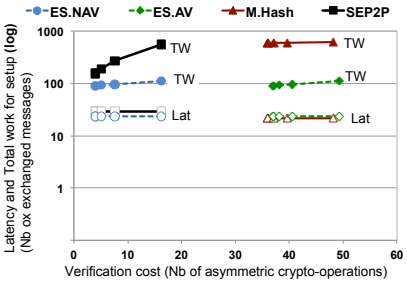

Figure 5: Setup communication costs

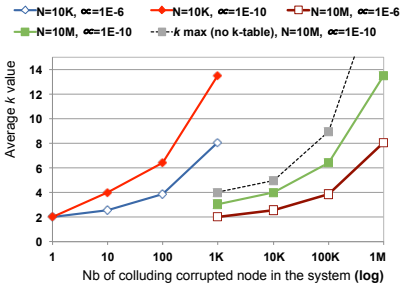

Figure 6: k versus C (N and $\propto$ vary)

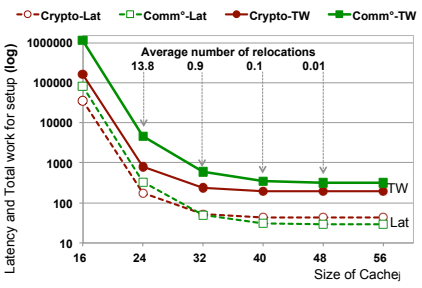

Figure 7: Setup costs varying R3 size

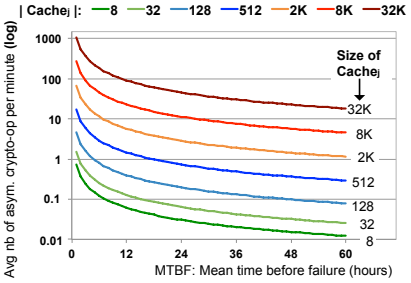

Figure 8: Maintenance overheads

run our protocol on k SL nodes thus increasing the total setup cost; and (2) we voluntarily make most of the checks during the setup (e.g., checking the actor certificates or verifying their availability) in order to reduce, as much as possible, the subsequent verification cost. Since this verification process will potentially be performed by a (very) large set of nodes (e.g., data sources), it is in our best interest to reduce it to avoid overloading the entire system. Figures 4 and 5 illustrate this aspect: our non-optimal setup cost is balanced by an optimized verification cost (and ideal disclosure in Figure 3). Note also that most operations are done in parallel (either by k *TLs* or *SLs*), thus leading to a reasonable setup latency (around 20 crypto-operations and 30 exchanged messages). We can also note in Figure 5 that M.Hash achieves the worst total work for setup (exchanged messages), because of the A routings in the DHT. Finally, we can remark the almost identical latency of ES.NAV, ES.AV and M.Hash on both metrics. Indeed, they all run the same initial protocol to compute RND_T. With respect to communication, the results are also identical because all DHT routings for M.Hash are done in parallel.

4.3 Scalability and Robustness

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

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

Number of actors: We also studied the impact of the variation of the number of actors. Overall, this results in a linear increase in the total work in terms of communications as the k *SLs* must check for the availability of A legitimate nodes to construct their respective CLs. For the sake of brevity, we omit here the detailed results.

Node cache size: We now focus on adapting the node cache size to the maximum number of required actors. Our goal is to evaluate the impact of the cache size on the global performances. To do so we take a reference network with $N = 100K$, $C\% = 1\%$ and $A = 32$ and vary the average cache size on the whole network (we compute $rs3$ easily dividing the cache size by N). Figure 7 shows the results (Y axis in log-scale). For each cache size, we simulated an execution on each node of the network and computed the average values for our metrics. Our measures show that with a very small cache, the probability of relocating the actor selection process is high (the *SLs* do not find enough legitimate

nodes in their cache w.r.t. R_3), which then leads to an increased latency and total work. Choosing a cache size greater than A, the query is almost never relocated (see Figure 7), giving better performances. This would lead to choose the largest possible cache. However, constructing such a cache also means maintaining it.

Maintenance costs: We also evaluated the impact of the cache size in the presence of node disconnections and, more generally, the impact of disconnections. To observe it, we simulated disconnections and measured their cost depending on the size of the node cache ($Cache_j$) using the default values for C, N, α and resulting k. We then considered those costs as a baseline and computed the global impact in a network where nodes disconnect (and reconnect) every x hours (mean time before failure or MTBF). We represent this cost in terms of asymmetric cryptographic operations (see Figure 8 - Y axis in log scale). The number of exchanged messages is not shown because graphs are very similar. We also computed these metrics for large node cache sizes (up to $32K$) to confirm that full mesh networks cannot be an alternative to DHT. Our results show that an overestimated cache is excessively costly even with an MTBF of 5 days: it consumes a large portion of the overall computing power of the entire system just to maintain it up to date. With small MTBFs, the network would be probably not maintainable. Since the number of actors for a computation is likely to be relatively small (e.g., few hundred, see Section 5), we can safely set the node cache size around 512 which leads to a reasonable maintenance cost (less than 1 signature per node per minute on average for MTBF = 1 day) and never trigger relocations (see Figure 7).

5 TASK ATOMICITY

5.1 Proposed Use Cases

We now focus on requirement 2, illustrating task atomicity on the use cases proposed in Section 1.

Use case 1: Mobile participatory sensing is used in many smart city applications for urban monitoring such as traffic monitoring (e.g., Waze or Navigon), evaluating the quality of road infrastructures, finding available parking spaces or noise mapping [36]. In these scenarios, the community members act as mobile probes and contribute to spatial aggregate statistics (density, averaged measures by location and time, spatial interpolation [36]) which in turn, benefit the whole community. As an alternative to the classical centralized architecture, the distributed PDMS paradigm increases the privacy guarantees for the users, thus encouraging their participation. A mobile user can generate sensing data (e.g., using her smartphone or vehicular systems) which is securely transmitted and recorded into her PDMS (e.g., a home box). This way each PDMS becomes a potential data source in the system. These data can then be aggregated by a small subset of data processor nodes to produce the required spatial aggregate statistics, which can be broadcasted to all the participating nodes.

Use case 2: Users can **subscribe to information flows based on their preference or user profile** (e.g., RSS feeds, specific product promotions or ads, etc.). A user profile can be represented by a set of concepts associating metadata terms (e.g., location, age, occupation, income, etc.) to values specific to each user. These associations are traditionally stored at a publication server to allow targeting the interested nodes. Instead, we propose to distributively store and index those profiles in SEP2P, thus greatly improving users' privacy. We call a concept index, an index associating for each concept the list of node addresses having this concept. Storing and searching this concept index is straightforward with a DHT. Each node does a $store(concept, IPaddress)$ for each concept in its profile. To find all the nodes matching a certain target profile (e.g., a logical expression of concepts), a DHT search is launched for each concept in the profile. Then, a set of randomly selected data processors are used to pick up the scattered pieces of the concept index, apply the logical expression of the target profile and compute the matching target nodes (TN), i.e., their IP addresses. Finally, the information is sent to the selected targets.

Use case 3: We consider **queries over the personal data contributed by a large set of individuals**, e.g., to compute recommendations, make participative studies. To achieve a high degree of pertinence and avoid flooding the system, such queries should target only a specific subset of the nodes, i.e., the nodes exposing a given user profile. Query examples are numerous, e.g., get the top-10 ranked movies by academics from Paris, or find the average number of sick leave days of pilots in their forties. The query processing is done in two steps which roughly correspond to the use case 2 combined with use case 1. First, the relevant subset of nodes, which match the query profile, must be discovered (use case 2). Then, the selected subset of target nodes become data sources which supply the required data (e.g., number of sick leave days) to compute the query result (use case 1). The main differences are that only the selected nodes provide data and that the result is transmitted only to the querier node and not to the entire system.

5.2 Detailed Node Roles

From the above description, we can define new node roles:

Node role 4. A **metadata indexer (*MI*)** stores part of the metadata shared by the nodes, allowing pertinent and efficient distributed data processing.

Node role 5. A **target finder (*TF*)**, applies a logical expression on its input to produce a list of target nodes.

Node role 6. A **data aggregator (*DA*)** applies an aggregative function to its input and produces partially aggregated results.

Node role 7. A **main data aggregator (*MDA*)** aggregates its input and produces the final result.

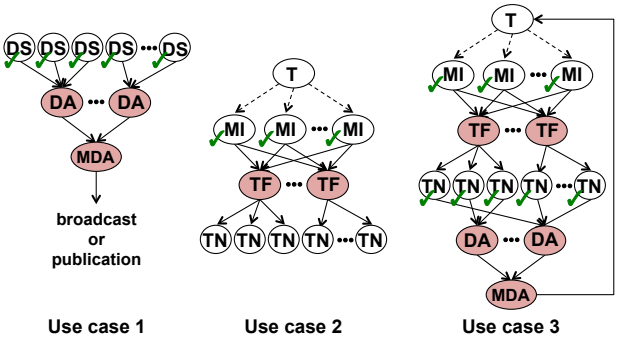

Figure 9: Distributed execution plans for the use cases

These roles allow designing distributed execution plans for the three use cases as shown in Figure 9. The nodes that must be chosen using the SEP2P protocol are shown in pink, and we used the symbol √ to denote that a node is a verifier (as specified in Section 3.6). This must be done each time a node discloses some sensitive data, thus on data sources and metadata indexers.

5.3 Towards Task Atomicity

The node roles and DEP proposed above already provide some task compartmentalization dividing the whole processing in tasks. However, much more can be done to minimize the impact of data leakage. In this section we present a few methods to achieve task atomicity. Our objective is mainly to show that task atomicity can be indeed performed and that it can significantly improve the system security when used in conjunction with the secure random actor selection. Given the space limitation, a detailed study of task atomicity is left for future work.

Metadata index protection: The concept index design already exhibits some form of task atomicity: (1) it is evenly distributed among all the nodes using the DHT mechanisms; (2) the imposed location of nodes in the DHT (see Section 3.2) leads to a randomized association between concepts and *MI* nodes. Nevertheless, a single corrupted node could disclose all the index information it owns. Further security improvements can be obtained by splitting each concept into s shares using the Shamir's secret sharing technique [32] which requires knowing at least p ($p \leq s$) shares to reconstruct the secret. Disclosing a single concept will now require p colluding nodes randomly selected.

User data protection: We consider here sensed data in use case 1 or the result of queries performed on a single PDMS in use case 2. Considering several *DA*s already reduces the impact of potential data leakage by a corrupted *DA* node. A simple way to reduce further this impact is to realize the aggregation on anonymized data (e.g., average traffic speed without user identity) or data without semantics (e.g., averaging data, a salary for instance, without knowing its meaning) or even encrypted data (with deterministic encryption). Note also that aggregation is continuous in the mobile sensing use case and that selected *DA* node will change at each iteration.

User identity protection: User's PDMS actively participate in the DEP either by receiving information (use case 1) or queries (use case 3) or by sending information (use cases 2 and 3). They thus communicate with *DA* nodes or receive messages from *TF* nodes, both being potentially corrupted. The reception / transmission task should be "isolated" to make one more step towards task atomicity. This can be achieved using the notion of proxy-forwarder that we illustrate for the *TN*-*DA* communication in the use case 3. The *TN* (which is actually a data source) must transmit its local result (e.g., number of sick leave days) to the *DA* node. *TN* can choose randomly any node P in the system and send the data, encrypted with the public key of the *DA* (known from the Verifiable Actor List). P will receive this data and transmit it to the *DA*. Thus, *DA* will have the data without knowing the sender, while P will know the sender but not the data. Note that (1) *TN* has good reasons to choose randomly P since it is the most interested in protecting its data; (2) the probability that both *DA* and P to be colluding nodes is extremely low ($\approx (C/N)^2$); and (3) we could use several proxies, thus mimicking anonymization network techniques (e.g., Tor).

6 RELATED WORK

DHT security. Several works focus on DHT security [40] considering the following attacks: (i) Sybil attack: an attacker generates numerous false DHT nodes to outnumber the honest nodes. Introducing an (offline) certificate authority, is deemed to be among the most effective defenses against the Sybil attack [11]. (ii) Routing table poisoning (eclipse attack): an attacker attempts to control most of the neighbors of honest nodes to isolate them. According to [40] the best strategy against such attacks 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 [34]. (iii) Routing and storage attacks: Sybil and eclipse attacks do not directly impact the DHT, they are mainly necessary means for future attacks, like various denials of service (DoS). For instance, the objectives might be to prevent a lookup request from reaching its destination, denying the existence of a valid key, or impersonating another node to deliver false data. These DoS attacks are usually classified as routing and storage attacks and most of the mechanisms employed to negate them are based on redundancy: at the storage and routing levels [40]. Thus, none of these works consider the secure and efficient actor selection for distributed processing as in SEP2P.

Secure Multi-party Computation and differential privacy. Cryptographic protocols have been proposed to protect the users' privacy in distributed computations with a focus on data confidentiality enforcement in personal data aggregation. Examples of computations related to this work are personal time-series clustering [2], kNN similarity queries [17], and location-based aggregate statistics [28]. However, MPC raises major scalability issues which in practice limit such protocols to specific types of computations [31].

Although it yields interesting results in privacy protection [15], differential privacy generally requires a central trusted aggregating node and ad-hoc adaptations depending on the targeted queries. As we search to provide a generic framework and exclude having a central actor to avoid a single point of failure, both requirements cannot be met by differential privacy. Even though local differential privacy [13] tries to address our first requirement, the solutions offered until now are still not generic, while the pertinence or the quality of the results may still be problematic with some applications [13]. Also, differential privacy exhibits intrinsic limitations with applications requiring continuous data flow aggregation (e.g., such as mobile participatory sensing) because of temporal correlation between consecutive data batches [10].

Distributed data aggregation using secure hardware. To overcome the limitations of MPC or differential privacy, several works propose using secure hardware at the user-side. Several secure protocols have been proposed for SQL aggregation [37], spatio-temporal aggregation [36], top-k full-text search [21], or privacy-preserving data publishing [3]. SEP2P also considers a secure PDMS at the user-side but our attack model considers having many colluding nodes. Moreover, the focus in SEP2P is on the secure and efficient random node selection. Differently, existing work focus on data aggregation or publishing and consider that all the nodes in the network participate in the protocol with their data being thus complementary to SEP2P.

Secure server-centric approaches. The above cited solutions are based on fully-distributed (P2P) or hybrid architectures. Alternatively, one could envision a solution based on a secured centralized server [6]. However, this raises important issues. First, users are exposed to sophisticated attacks, whose cost-benefit is high on a centralized database. Second, centralizing all users' data into one powerful server makes little sense in the PDMS context in which data is naturally distributed at the users' side. Hence, users might be reluctant to use such a massively centralized data service. Finally, new legislation such as the European GDPR [27] may hinder the development of such centralized solutions.

7 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 private 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 important and challenging issue. To this end, we proposed SEP2P, a fully-distributed P2P system laying the foundation for secure, efficient and scalable execution of distributed computations. By considering a realistic threat model, we analyzed the fundamental security and efficiency requirements of such a distributed system. We showed that the secure selection of random actor nodes is the basis of security for any distributed computation. Then, we proposed secure and highly efficient protocols to address the actor selection problem. Our simulation-based experimental evaluation indicates that our protocol leads to minimal private information leakage, i.e., increasing linearly with the number of colluding nodes. At the same time, the cost of the security mechanisms depends only on the maximum number of colluding nodes and remains very low even with wide collusion attacks.

This work opens the way for several interesting research problems. In particular, to further minimize the impact of a private data leakage, one should complement our random actor selection with task atomicity, i.e., decompose the computation process such that it minimizes the amount of sensitive data the processor nodes have access to. To underline the importance of this requirement, we discussed in this paper three types of representative applications in the PDMS context and provided sketches of solutions to achieve task atomicity. Certainly, this problem deserves a deeper look and constitutes our main objective as future work.

REFERENCES

[1] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Yanli Guo, Lionel Le Folgoc, Benjamin Nguyen, Philippe Pucheral, Indrajit Ray, Indrakshi Ray, and Shaoyi Yin. 2010. Secure personal data servers: a vision paper. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 25–35.

[2] Tristan Allard, Georges Hébrail, Florent Masseglia, and Esther Pacitti. 2015. Chiaroscuro: Transparency and privacy for massive personal time-series clustering. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 779–794.

[3] Tristan Allard, Benjamin Nguyen, and Philippe Pucheral. 2014. METAP: revisiting Privacy-Preserving Data Publishing using secure devices. *Distributed and Parallel Databases* 32, 2 (2014), 191–244.

[4] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral, Iulian Sandu Popa, and Guillaume Scerri. 2019. Personal Data Management Systems: The security and functionality standpoint. *Information Systems* 80 (2019), 13 – 35.

[5] Nicolas Anciaux, Luc Bouganim, Philippe Pucheral, Yanli Guo, Lionel Le Folgoc, and Shaoyi Yin. 2014. MILo-DB: a personal, secure and portable database machine. *Distributed and Parallel Databases* 32, 1 (2014), 37–63.

[6] Arvind Arasu, Spyros Blanas, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, Prasang Upadhyaya, and Ramarathnam Venkatesan. 2013. Secure database-as-a-service with cipherbase. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 1033–1036.

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

[8] Michael Backes, Peter Druschel, Andreas Haeberlen, and Dominique Unruh. 2009. CSAR: A Practical and Provable Technique to Make Randomized Systems Accountable.. In *NDSS*, Vol. 9. 341–353.

[9] Blue Button. 2010. Find Your Health Data. (2010). Retrieved October 12, 2018 from https://www.healthit.gov/topic/health-it-initiatives/blue-button

[10] Yang Cao, Masatoshi Yoshikawa, Yonghui Xiao, and Li Xiong. 2017. Quantifying Differential Privacy under Temporal Correlations. In *33rd IEEE International Conference on Data Engineering, ICDE 2017*. 821–832.

[11] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S Wallach. 2002. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 299–314.

[12] Cozy Cloud. 2013. Your digital home. (2013). Retrieved October 12, 2018 from https://cozy.io/en

[13] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. 2018. Privacy at Scale: Local Differential Privacy in Practice. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018*. 1655–1658.

[14] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. 2014. openpds: Protecting the privacy of metadata through safeanswers. *PloS one* 9, 7 (2014), e98790.

[15] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. 3574–3583.

[16] Fing. 2013. The mesinfos project explores the self data concept in france. (July 2013). Retrieved October 12, 2018 from http://mesinfos.fing.org/english

[17] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Antoine Rault, François Taïani, and Jingjing Wang. 2015. Hide & Share: Landmark-based Similarity for Private KNN Computation. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE, 263–274.

[18] Javier González, Michael Hölzl, Peter Riedl, Philippe Bonnet, and René Mayrhofer. 2014. A practical hardware-assisted approach to customize trusted boot for mobile devices. In *International Conference on Information Security*. Springer, 542–554.

[19] Anne-Marie Kermarrec and François Taïani. 2015. Want to scale in centralized systems? Think P2P. *J. Internet Services and Applications* 6, 1 (2015), 16:1–16:12.

[20] Saliha Lallali, Nicolas Anciaux, Iulian Sandu Popa, and Philippe Pucheral. 2017. Supporting secure keyword search in the personal cloud. *Information Systems* 72 (2017), 1–26.

[21] Thi Bao Thu Le, Nicolas Anciaux, Sébastien Gilloton, Saliha Lallali, Philippe Pucheral, Iulian Sandu Popa, and Chao Chen. 2016. Distributed secure search in the personal cloud. In *19th International Conference on Extending Database Technology (EDBT 2016)*. 652–655.

[22] Sangmin Lee, Edmund L Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov. 2013. πBox: A Platform for Privacy-Preserving Apps.. In *NSDI*. 501–514.

[23] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*. Springer, 53–65.

[24] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. 1996. *Handbook of Applied Cryptography*. CRC Press.

[25] MiData. 2011. The midata vision of consumer empowerment. (2011). Retrieved October 12, 2018 from https://www.gov.uk/government/news/the-midata-vision-of-consumer-empowerment

[26] Nextcloud. 2016. Protecting your data. (Jun 2016). Retrieved October 12, 2018 from https://nextcloud.com

[27] European Parliament. 2016. General Data Protection Regulation. Law. (27 April 2016). Retrieved October 12, 2018 from https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679

[28] Raluca Ada Popa, Andrew J Blumberg, Hari Balakrishnan, and Frank H Li. 2011. Privacy and accountability for location-based aggregate statistics. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 653–666.

[29] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database using SGX. In *EnclaveDB: A Secure Database using SGX*. IEEE, 0.

[30] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001. *A scalable content-addressable network*. Vol. 31. ACM.

[31] Eyad Saleh, Ahmad Alsa'deh, Ahmad Kayed, and Christoph Meinel. 2016. Processing over encrypted data: between theory and practice. *ACM SIGMOD Record* 45, 3 (2016), 5–16.

[32] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[33] Solid. 2018. Solid empowers users and organizations to separate their data from the applications that use it. (2018). Retrieved October 12, 2018 from https://solid.inrupt.com/

[34] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.

[35] ARM Security Technology. 2008. *Building a Secure System using TrustZone Technology*. Technical Report. ARM.

[36] Dai Hai Ton That, Iulian Sandu Popa, Karine Zeitouni, and Cristian Borcea. 2016. PAMPAS: Privacy-Aware Mobile Participatory Sensing Using Secure Probes. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*. ACM, 4.

[37] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. 2016. Private and scalable execution of SQL aggregates on a secure decentralized architecture. *ACM Transactions on Database Systems (TODS)* 41, 3 (2016), 16.

[38] Jordi Creus Tomàs, Bernd Amann, Nicolas Travers, and Dan Vodislav. 2011. RoSeS: a continuous query processor for large-scale RSS filtering and aggregation. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management*. 2549–2552.

[39] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. 2011. A survey of DHT security techniques. *ACM Computing Surveys (CSUR)* 43, 2 (2011), 8.

[40] Qiyan Wang and Nikita Borisov. 2012. Octopus: A secure and anonymous DHT lookup. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 325–334.