A Data Privacy Service for Structured P2P Systems

Mohamed Jawad, Patricia Serrano-Alvarado, Patrick Valduriez, Stéphane Drapeau

A Data Privacy Service for Structured P2P Systems

Mohamed Jawad, Patricia Serrano-Alvarado
LINA, University of Nantes
Nantes, France
Email: Name.Lastname@univ-nantes.fr

Patrick Valduriez
INRIA and LIRMM
Montpellier, France
Email: Patrick.Valduriez@inria.fr

Stéphane Drapeau
Obeo
Rezé, France
Email: Stephane.Drapeau@Obeo.fr

***Abstract*—Online peer-to-peer (P2P) communities such as professional ones (e.g., medical or research) are becoming popular due to increasing needs on data sharing. P2P environments offer valuable characteristics but limited guarantees when sharing sensitive or confidential data. They can be considered as hostile because data can be accessed by everyone (by potentially untrustworthy peers) and used for everything (e.g., for marketing or for activities against the owner's preferences or ethics). In this paper we propose PriServ, a privacy service located on top of distributed hash table (DHT) based P2P systems which prevents data privacy violations. Based on data owner privacy preferences, PriServ uses Hippocratic database principles, takes into account which operations will be realized on shared data (e.g., read, write, disclosure) and uses reputation techniques to increase trust on peers. Several simulation results encourage our ideas and a prototype of PriServ is under development[1].**

***Keywords*-Peer-to-peer; DHT; Data privacy; Purpose based access control; Trust;**

I. Introduction

In our days, the democratization of the use of information systems and the massive data digitalization allow to identify all aspects of a person's life. For instance, their professional performance (e.g., publish or perish software, dblp website), their client's profile (e.g., thanks to fidelity smart cards), their user's profile (e.g., thanks to their user identity, access localities, generated traffic) or their health level (e.g., thanks to the digitalization of medical records). Those observations have risen serious data privacy concerns.

Online peer-to-peer (P2P) communities such as professional ones (e.g., medical or research) are becoming popular due to increasing needs on data sharing. In this context, P2P environments offer valuable characteristics (e.g., *scalability, distribution, autonomy*) but limited guarantees concerning data privacy. They can be considered as hostile because data, that can be sensitive or confidential, can be accessed by everyone (by potentially untrustworthy peers) and used for everything (e.g., for marketing, profiling, fraudulence or for activities against the owner's preferences or ethics).

Data privacy is the right of individuals to determine for themselves *when*, *how* and *to what* extent information about them is communicated to others [26]. It has been treated by many organizations and legislations which have defined well accepted principles. According to OECD[2], data privacy should consider: collection limitation, purpose specification, use limitation, data quality, security safeguards, openness, individual participation, and accountability. From these principles we underline *purpose* specification which states that *data owners* should be able to specify the data access objective for which their data will be collected and used.

Several solutions that follow the OECD guidelines have been proposed. A major solution is Hippocratic databases which enforces purpose-based disclosure control in a centralized relational datastore [1], [2], [5], [17]. This is achieved by using privacy metadata, i.e. privacy policies and privacy authorizations stored in tables. A privacy policy defines (for each attribute, tuple or table) the usage purpose(s), the potential users and retention period while privacy authorization defines which purposes each user is authorized to use. In a Hippocratic database, queries are submitted to the database along with their intended purpose. Query execution preserves privacy using query modification and restrictions by column/row/cell.

In addition to purpose-based data privacy, to prevent data misuse, it is necessary to trust participants. Trust management systems deal with unknown participants by testing their reputation [15], [18], [19], [24]. Reputation techniques verify the trustworthiness of peers by assigning them *trust levels*. A trust level is an assessment of the probability that a peer will not cheat. A peer with a high trust level is considered as honest and a peer with a low trust level as malicious. Trust levels are updated (incremented/decremented) in order to reflect peers' behavior. If a peer misbehaves, its trust level is decremented.

A. Motivations.

In the context of P2P systems, few solutions for data privacy have been proposed and they focus on a small part of the general problem of data privacy, e.g. *anonymity* of uploaders/downloaders, *linkability* (correlation between uploaders and downloaders), *content deniability*, *data encryption* and *authenticity* [6], [16], [21], [25]. However, the major problem of data privacy violation due to data disclosure to malicious peers which misuse data, is not addressed.

As a motivating example, consider a collaborative medical research focusing on the evolution of cardiovascular diseases (e.g. heart attacks, atherosclerosis, etc.) depending on patient characteristics (e.g. age, diet, smoking, sports, gender, etc.). The participants of this research are scientists, doctors, patients, pharmacist, nurses, medical students, etc. In order to control disclosure on sensitive data (e.g., medial records owned

[1]Work partially funded by the DataRing project of the french ANR.

[2]Organization for Economic Cooperation and Development. One of the world's largest and most reliable source of comparable statistics, on economic and social data. http://www.oecd.org/.

by doctors or research results owned by scientists) without violating privacy, data access should respect the privacy preferences of data owners. They can be defined in the following manner:

- A doctor may allow *reading* access on her medical records to a *patient* for *seeing her diagnosis*.
- A doctor may allow *reading* access on information such as age but do not to information such as name or social security number, to *scientists* for *researching* on the evolution of the cardiovascular disease.
- A doctor may allow *writing* access on her information to *researchers* for *adding comments* on her diagnosis.
- A researcher may allow *reading* access on her research results to *doctors* for *diagnosing*.

In this P2P application, sharing data based on data owners privacy preferences is a challenge. Besides ensuring that data disclosure should be done only to specified participants, purposes (e.g. seeing one's diagnosis, researching, etc.) and specified operations (e.g., writing, reading) defined by data owners, should be respected by data requesters. In addition, data should not be shared indistinctly with all participants (all scientists or all doctors in the system). It is necessary to consider the concept of trust among participants. For instance, doctors need to trust each researcher to share their private data with her.

Currently, publishing and requesting data in P2P systems do not take into account privacy preferences. Thus, controlling data sharing, with trustworthy peers, for specific purposes and operations, is not possible in such systems without adding new services. In this context, an efficient P2P purpose-based privacy service with trust control is needed.

B. Contributions.

This paper has two main contributions.

- We propose a P2P data privacy model. The goal of such model is to provide the basis of a framework that facilitates the prevention of data owner's privacy violation by 1) allowing them to specify their privacy preferences and by 2) restraining data users to specify the purpose and the operation for which they request data.
- We propose PriServ[3], a privacy service on top of DHT which, based on the proposed model, prevents privacy violation by limiting malicious data access. For that, we use purpose and operation based access control as well as trust techniques. To our knowledge, PriServ is the first proposition that introduces data access based on purposes in P2P systems. The performance evaluation of our approach through simulation shows that the overhead introduced by PriServ is small.

In the following, Section II presents the P2P data privacy model we propose, Section III presents PriServ, Section IV presents performance evaluation, Section V discusses related work and section VI concludes this paper.

[3]In [13], [14], some aspects of this work were proposed.

II. P2P Data Privacy Model

We consider that peers misbehave if they violate data privacy preferences defined by data owners. There exist numerous malicious behaviors when sharing data, in this model we concentrate on preventing:

- **Unauthorized disclosure.** Server peers can misbehave by disclosing data to peers who, based on data privacy preferences, should not have those data.
- **Data misuse.** Requester peers misbehave if they do not respect agreements made during data access.
- **Attacks to data integrity.** Server peers may violate data integrity by modifying data content of data they provide without permission.

To prevent those misbehaviors, the proposed P2P data privacy model uses mainly Hippocratic principles (purpose-based access control) and trust notions (reputation techniques). This section introduces some necessary concepts (Section II-A), the used data model (Section II-B) and finally the functions of the model (Section II-C).

A. Basic Concepts

In this section we present the basic concepts of our P2P data privacy model.

1) P2P Systems: A P2P system is a distributed system composed of participants (peers) with similar characteristics, that can take the role of "clients" and "servers". It can scale up to a large number of participants and allows sharing large number of data.

A peer represents a participant who shares data, requests information, or simply contributes in the execution of a service or a query. We consider also that there exist peers who own data and that do not necessarily act as servers of those data. A peer can join or leave the system at any time without restriction. Thus we distinguish three types of peers:

- **Owner.** A peer that owns and shares data.
- **Requester.** A peer that requests data.
- **Server.** A peer that provides data.

The present model makes no assumptions about the type of P2P system, it can be structured or unstructured. We only consider that a peer has a unique identifier in the system.

2) Privacy Policies: Data privacy preferences of data owners are registered in privacy policies (PP). Inspired from P3P [7], Figure 1 shows a PP model. This model does not claim to be exhaustive. It shows information about PPs which can include:

- **Authorized users.** A list of users who are authorized to access data. A user can be an individual or a group. In a P2P system, users are hardly known in advance. In the following, to simplify, we will consider that individual users belong to groups.
- **Operations.** An operation determines what peers can do with data. We use three basic operations, read, write and disclose (other operations can be defined).
  - **Read.** A peer can read the data content.

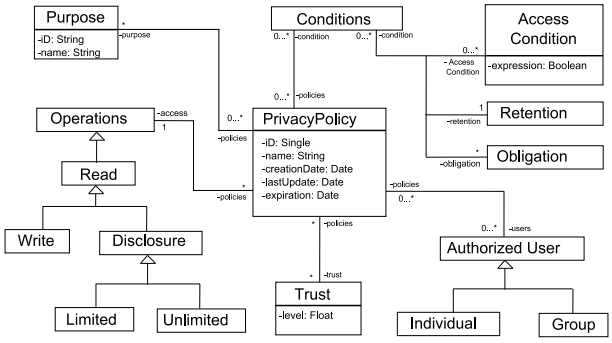


Fig. 1. Privacy policy (PP) model

  - **Write.** A peer can modify (insert, update, delete) the data content.
  - **Disclose.** A peer is able to disclose shared data. Disclosure can be limited to a predefined time duration.
- **Purpose.** It states the data access objective. Data owners should be able to specify the purposes for which users can access their data.
- **Access conditions.** They state under which semantic conditions data can be accessed. This may concern data values, for example age>10.
- **Retention time.** It states the limited retention time of data. For example, the record of patient x should be used only during two months.
- **Obligations.** They state the tasks a user must accomplish after the data access, for example, researcher Ri should return research results involving the record of patient x.
- **Minimal trust level.** It is the minimal trust level a requester peer should have in order to gain data access.

3) Trust: It is a fuzzy concept where trustworthy peers are supposed to respect privacy policies defined by data owners. Trust between peers is important in order to control data access.

- **Reputation.** It is an overall estimation of the behavior of a peer. This estimation depends on the interaction of peers.
- **Trust level.** The trust level reflects a peer reputation with respect to other peers. A peer can have different trust levels at different peers. Trust levels vary in a range of [0,1].
  - An **honest peer** is a peer which has a high trust level e.g., [0.5, 1].
  - A **malicious peer** is a peer which has a low trust level e.g., [0, 0.5).
  - An **unknown peer** is a peer which has an unknown trust level.

  Peers can have locally the trust levels of some peers. If a peer does not have a particular trust level, it can ask for it to other peers called *friends*.
- **Friend.** A friend of a peer P is a peer with a high trust level from P's point of view. The number of friends held by a peer can vary from one peer to another.

B. Data Model

In order to respect PPs, we define a specific data model where PPs are associated with data. In this paper, we use relational tables. However our model can consider any type of data (files, XML documents, rich text files, etc.).

1) Data Tables: We consider that each owner peer stores locally the data it wants to share. Those data are stored in relational tables which we call *data tables*. One important restriction on these tables is that primary keys should be generic and impersonal in order to avoid disclosure of personal information. Following our motivating example, Table I shows medical records of doctor Dj.

2) Privacy Policies Table: Data contained in PPs are stored in a table named *privacy policies table*. Table II shows the privacy policies table of doctor Dj. To simplify, we do not include all elements of Figure 1. In this table, one tuple corresponds to one PP. The same PP can be applied to different data. Each policy contains operations (read, write, disclose), allowed users, access purposes, conditions (if they exist), and the required minimal trust level of allowed users.

3) Private Data Table: This table joins the data and the privacy policies tables. Each tuple defines the data subject to privacy (table, column or line) and the corresponding privacy policy id (PPID). Table III shows the private data table of doctor Dj. This table allows cell-based access control if the table, column and line are specified. In PD1, only some columns of the data table DTj (those who do not disclose patients identities) are concerned by the privacy policy PP1 where pharmacists and doctors are allowed to read records of patients who were born before 2000. In PD3 and PD4 the diagnosis cell of Pat3 and Pat4 can be modified by doctor Dk. It is assumed that doctor Dk is concerned also by PD1 so before modifying the diagnosis she can read the complete patient record.

More sophisticated ways of attaching data to privacy preferences can be used but they go beyond the goal of this work.

C. Basic functions

The basic operations proposed in this model are publishReference(), publishData() and requesting().

a) Boolean publishData(data, PPId): Owner peers use this function to publish data content in the system. The second parameter is the privacy policy that dictates the usage conditions and access restrictions of the published data. This function returns true if data content is successfully distributed, false otherwise. To protect data privacy against potential untrusted servers, before distribution, data content is encrypted (symmetric cryptography).

b) Boolean publishReference(data, PPId): Owner peers use this function to publish data references in the system. The second parameter is the privacy policy that dictates the usage conditions and access restrictions of the published data references. This function returns true if data references are successfully distributed, false otherwise. Servers store data references and help requesters to find data owners to obtain data content. Publishing only data references allows owners

Data table DTj							
Id (PK)	**SS**	**Name**	**Country**	**Birthdate**	**Gender**	**Smoker**	**Diagnosis**
Pat1	001044001001	Alex	France	2000	Male	No	NO cardiovascular disease
Pat2	900344001001	Bea	France	1990	Female	No	NO cardiovascular disease
Pat3	730844001001	Chris	France	1973	Male	No	YES cardiovascular disease
Pat4	441144001001	Dave	Belgium	1944	Male	Yes	YES cardiovascular disease
Pat5	680544001001	Elena	Russia	1968	Female	Yes	YES cardiovascular disease

TABLE I
DATA TABLE OF DOCTOR DJ

Privacy policies table j					
Id (PK)	**Operation**	**User**	**Purpose**	**Condition**	**Minimal trust level**
PP1	Read	Pharmacist, Doctors	Consulting record	Birthdate $<$ 2000	0.5
PP2	Read	Researchers	Researching on cardiovascular disease	—	0.6
PP3	Write	Dk	Second diagnosis	—	0.9

TABLE II
PRIVACY POLICIES TABLE OF DOCTOR DJ

Private data table j				
Id (PK)	**Data**			**Privacy Policy**
	Table	**Column**	**Id**	
PD1	DTj	Birthdate, Gender, Smoker, Diagnoses	—	PP1
PD2	DTj	Country, Birthdate, Gender, Smoker, Diagnosis	—	PP2
PD3	DTj	Diagnosis	Pat3	PP3
PD4	DTj	Diagnosis	Pat5	PP3

TABLE III
PRIVATE DATA TABLE OF DOCTOR DJ

to publish private data while being sure that data content will be provided to the right requesters. This hypothesis can not be guaranteed in the previous function because servers may misbehave by returning encrypted data to unauthorized peers.

c) Data request(dataRef, purpose, operation): Requester peers use this function to request data (*dataRef*) for a specific purpose (e.g., researching, diagnosis, analyzing) to perform a specific operation (i.e., read, write, disclosure). This function returns the requested data if the requester has corresponding rights, otherwise it returns null. This function compels requesters to specify the access purposes and the operation that they will apply to requested data. This explicit request commits users to use data only for specified purposes and operations. Legally, this commitment, may be used against malicious users if data are used for other purposes/operations.

As we will see in the next section, during the execution of the requesting() function, the trust level of requesters is verified. Next section introduces PriServ, the service implementing this model.

III. PRISERV DESIGN

PriServ is an implementation of the proposed privacy model. It provides the publishReference(), publishData() and requesting() functions. It is implemented on top of a DHT-based P2P system where only the get() and put() functions are used. Next section introduces our design choices, Section III-B presents the PriServ architecture and Section III-C an analysis of incurred costs based on the number of messages.

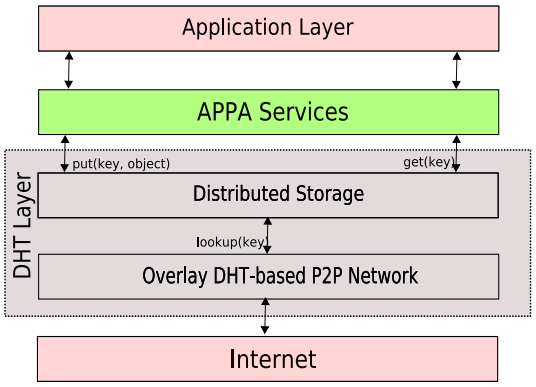


Fig. 2. Global architecture

A. Design Choices

In this section, we present the DHT-based architecture used by PriServ and the way the data keys, managed by the DHT layer, are defined.

1) DHT-based P2P architecture: All DHT systems (e.g. Chord [23], Pastry [22], etc.) support a distributed lookup protocol that efficiently locates the peer that stores a particular data item. Data location is based on associating a *key* with each data item, and storing the key/data item pair at the peer to which the key maps. To generalize, DHT provides two basic operations [8], each incurring $O(logN)$ messages.

- $put(k, data)$ stores a key k and its associated data object

in the DHT.
- $get(k)$ retrieves the data object associated with k in the DHT.

Figure 2 shows the considered global architecture. On top of the Internet network there is the P2P system. The overlay network layer takes in charge the routing system by implementing the *lookup()* function but also by managing the peers dynamicity (join/leave of peers). On top of this layer, the distributed storage layer ensures key-based data searching and data distribution by implementing the put() and get() functions. Those two layers make abstraction of the DHT-based P2P system. PriServ is implemented as an APPA (Atlas P2P Architecture) service [3] on top of the DHT layer. The PriServ implementation uses Chord for its efficiency and simplicity. Nevertheless, any DHT-based P2P system can be used.

In Chord, a DHT maps a key *k* to a peer *P* called *responsible for k with respect to a hash function h*. Peers maintain information about $O(logN)$ other peers in a *finger table* and resolve lookups via ***O(logN)*** messages to other peers. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant peer. A consistent hash function assigns to each peer and key an m-bit *identifier* using a hash function such as SHA-1 [9]. A peer identifier is chosen by hashing the peer IP address. A key identifier is based on data values that can be a data reference, an address, etc. All peers are ordered in a circle modulo 2^m. Key k is assigned to the first peer whose identifier is equal to or follows k in the identifier space. This peer is called the $successor$ of k. Chord provides data publishing and searching using only ***O(logN)*** messages.

2) Data keys: We consider that the DHT generates peer identifiers by hashing the peer IP address (like in Chord). Concerning data keys, to enforce data privacy, we propose to hash the triplet (*dataRef, purpose, operation*). *dataRef* is a unique data reference, *purpose* is the data access purpose and *operation* is the operation that will be executed on requested data. Thus, the same data with different access purposes and different operations have different keys[4].

Because keys contain the notion of purposes and operations, requesting data is always made for a defined purpose and operation. This allows to enhance access control.

Data references are expressed as follows. Inspired by PIER [12], we propose to concatenate the data table name, the column name and the value of the primary key. By using this last value, the searching granularity can be a tuple. If shared data concerns an entire table, the column name and the value of the primary key are omitted. Similarly, if a particular column is concerned, the value of the primary key is omitted. If several columns of the same table are concerned they can be enclosed in curly brackets. For instance, in table III, the data reference for PD1 is DTj.{Birthdate, Gender, Smoker, Diagnoses} and for PD3 is DTj.Diagnosis.Pat3.

[4][10] has analyzed how to manage schemas and process simple queries efficiently in a flexible P2P environment. Thus, we consider that all peers accessing the same data are capable of schema mapping and that peers allowed to access particular data are informed of their allowed purposes. Thus, requester peers are able to produce keys.

B. PriServ architecture

Figure 3 shows the PriServ architecture. Publishing and requesting are always made through PriServ. Principal components of the PriServ service (storage manager, policy manager, key manager, cipher manager, data signature manager, trust manager) are organized by an orchestrator. In this section we present those components and the algorithms where they are used.

1) Components:

a) Key manager: Its role is to generate data keys. It creates data keys by hashing data references, purposes and operations. This component offers two types of functions. The first one, used in publishReference() and publishData(), returns the created data key. The second, used in request(), returns the created data key and attaches it the requester identifier in order to identify the requester during the requesting process.

b) Policy manager: Its role is to manage PPs. Data owners organize their privacy preferences in PPs (see Figure 1) which are stored by the policy manager in Table II. From PPs, this component creates data usage conditions (see conditions in Figure 1) that should be respected by requesters. From PPs, this component also extracts the list of authorized users. This list is a kind of ACL (Access Control List) that contains only allowed users. We recall that in this work, for simplification, we consider that users belong to groups. Thus, this ACL contains mainly a list of groups of users and maybe some individual users known in advance.

c) Trust manager: Its role is to manage trust levels. We consider that each peer stores locally a list of peers and the corresponding trust levels. If a required trust level does not exist locally, the trust manager asks for it to other peers. In PriServ three ways of obtaining trust levels are used, namely, with-friends, without-friends and with-or-without-friends (see Section III-B2d). The use of the trust manager allows owners to increase their data privacy protection level. However, the use of the trust manager is optional because it may generate high costs.

d) Storage manager: Its role is to manage data storage. Stored data are mainly Tables I and III. Data storage is made locally before data are stored in the P2P system. To store data in the P2P system, this component invokes the put() and get() operations of the DHT layer.

e) Cipher manager: Its role is to manage cryptography in PriServ. It offers a function that creates a symmetric cipher key for each pair (data, PP). It also offers two functions to encrypt and decrypt data. PriServ is independent of the encryption technique used to encrypt data.

f) Data signature manager: Its role is to manage data signatures to check the integrity of data. This component offers a function to calculate digital data signatures (e.g., MD5). PriServ is independent of the technique used to create data signatures.

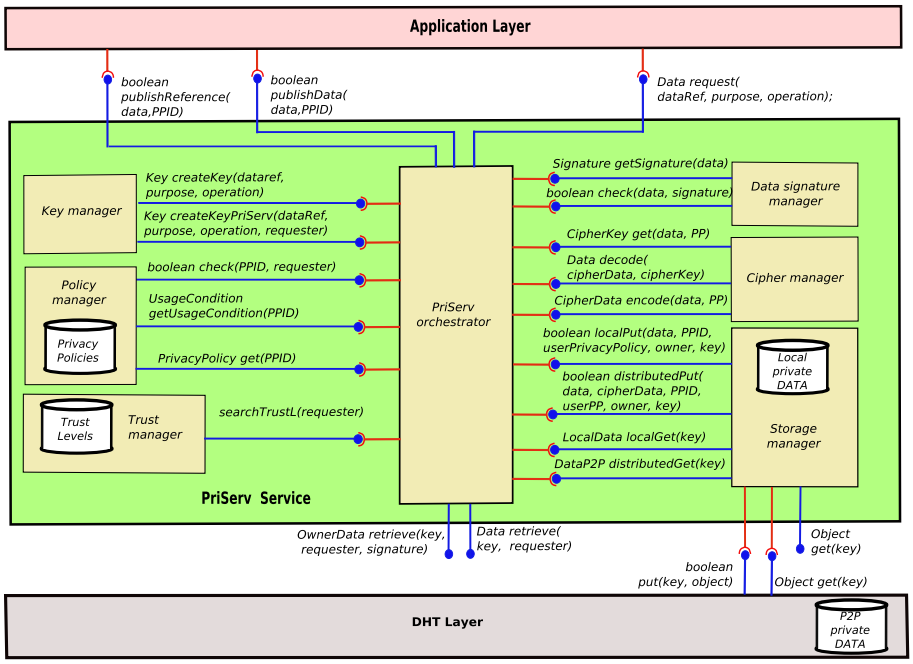


Fig. 3. PriServ architecture

g) PriServ orchestrator: It is the central component of PriServ. According to the peer type (i.e., requester, owner, server), the orchestrator executes a different workflow by using the components introduced before.

- ***Owner orchestrator.*** Its role is to orchestrate the owner functionalities. It is responsible of publishing owner references or data depending on the called function (publishData() or publishReference()). It is also responsible of retrieving data or symmetric keys during the requesting process. It interacts with the application layer during publishing and with the requester orchestrator during retrieving.
- ***Requester orchestrator.*** Its role is to orchestrate data requesting. It interacts with the application layer during requesting and with the owner orchestrator during retrieving.
- ***Server orchestrator.*** Its role is to orchestrate the server functionalities. For that, it interacts with the DHT layer to store and retrieve data of the P2P system.

2) Functions: This section presents the publishing and requesting algorithms implemented by PriServ and gives an overview of the trust level searching algorithms presented in [14].

a) PublishReference(data, PPID): When the owner orchestrator receives this call, it uses the algorithm shown in Figure 4 in order to publish data references in the P2P system and to store data locally. In line 0, the owner application sends the privacy policy identifier (PPID) and the data to publish. In lines 2 and 3, the owner orchestrator asks the policy manager the PP and the usage conditions of PPID. Then, it asks the key manager to create the data key in line 4. Finally, the orchestrator asks the storage manager to locally put the data in line 5.

```
Owner orchestrator
0: publishReference(data, PPID)
1: begin
2:   privacyPolicy = policyManager.get(PPID);
3:   userCondition =
                  policyManager.getUserCondition(PPID);
4:   key = keyManager.createKey(data.dataref,
                                privacyPolicy.purpose,
                                privacyPolicy.operation);
5:   storageManager.localPut(data, PPID, userCondition,
                              privacyPolicy.ACL,
                              owner, key);
6: end;

Owner storage manager
7: localPut(data, PPID, userCondition, ACL, owner, key)
8: begin
9:   DataTable.localSave(data);
10:  PrivateDataTable.localSave(data.dataRef,PPID);
11:  dataP2P = createDataP2P(userCondition, ACL,
                             owner);
12:  DHT.put(key,dataP2P);
13: end;
```

Fig. 4. PublishReference() function

In lines 9 and 10, the storage manager stores locally the data in the data table (Table I), then the data reference and the PPID in the private data table (Table III). In line 11, it creates the P2P data that will be stored on the P2P system. P2P data is composed of the owner identifier, the usage condition and the ACL. Finally, in 12, it invokes the DHT to put the P2P data on the P2P system.

b) PublishData(data, PPID): When the owner orchestrator receives this call it uses the algorithm shown in Figure 5 to publish the data content in the P2P system and to store

```
Owner orchestrator
0: publishData(data, PPID)
1: begin
2:   privacyPolicy = policyManager.get(PPID);
3:   userCondition =
                policyManager.getUserCondition(PPID);
4:   key = keyManager.createKey(data.dataref,
                              privacyPolicy.purpose,
                              privacyPolicy.operation);
5:   cipherData = cipherManager.encode(data,
                                       privacyPolicy);
6:   storageManager.distributedPut(data , PPID,
                                   cipherData,
                                   userCondition,
                                   privacyPolicy.ACL,
                                   owner, key);
7: end;

Owner storage Manager
8: distributedPut(data , PPID, cipherData,
                  userCondition, ACL, owner, key);
9: begin
10:  DataTable.localSave(data);
11:  PrivateDataTable.localSave(data.dataRef,PPID);
12:  dataP2P = createDataP2P(cipherData,
                              userCondition,
                              ACL, owner);
13:  DHT.put(key,dataP2P);
14: end;
```

Fig. 5. publishData() function

```
Requester orchestrator
0: Data request(dataRef, purpose, operation)
1: begin
2:   data = null;
3:   key = keyManager.createKeyPriServ(dataRef, purpose,
                                        operation, requester);
4:   dataP2P = storageManager.distributedGet(key);
5:   if (dataP2P contains cipher data) do
6:     signature = dataSignatureManager.getSignature(
                                        dataP2P.cipherData);
7:     ownerData = dataP2P.owner.retrieve(key,
                                          requester,
                                          signature);
8:     if (ownerData contains a cipherKey) do
9:        data = cipherManager.decode(
                                    dataP2P.cipherData,
                                    ownerData.cipherKey);
10:     else if (ownerData contains data) do
11:           data = ownerData.data;
12:         endIf;
13:    endIf;
14:  endIf;
15:  if (dataP2P contains a data reference) do
16:      data = dataP2P.Owner.retrieve(key, requester);
17:  endIf;
18:  return data;
19: end;

Requester storage manager
20: dataP2P distributedGet(key)
21: begin
22:   dataP2P = DHT.get(key);
23:   return dataP2P;
24: end;
```

Fig. 6. request() function

locally a copy of the data. Lines 2 to 4 are the same as in the publishReference() function (same lines). In order to put data on the P2P system, data should be encrypted, the orchestrator contacts the cipher manager to obtain the cipher data in line 5. Finally, in line 6, the orchestrator asks the storage manager to store locally the data then to put them on the P2P system. Notice that in this step, cipher data are transmitted to the storage manager.

Lines 10 and 11 executed by the storage manager are the same as in the publishReference() function (lines 9 and 10). In line 12, the storage manager creates the P2P data to store on the P2P system which is composed of the cipher data, the owner identifier, the usage condition and the ACL. Finally the storage manager invokes the DHT to put the P2P data on the P2P system.

It is important to underline that the DHT is used in the same way in both publishing functions, lines 12 in the first function and 13 in the second one. Only the content of the data sent is different.

c) Requesting: In PriServ, data requesting is done in three steps: data requesting to the server, data retrieving from the owner and a trust level searching made by the owner.

Request(dataRef, purpose, operation): When a requester orchestrator receives this call it uses the algorithm shown in Figure 6. In line 3, the requester orchestrator asks the key manager to create the data key. Then, it asks its storage manager to get the data from the P2P system in line 4. The storage manager contacts the server storing the data corresponding to the created data key by invoking the get() function of the DHT.

When a server peer receives a get message it checks if the requester is authorized to access the data by using the ACL attached to each data. Then it returns the stored P2P data to the requester storage manager in line 22. Finally, the storage manager returns the P2P data to its orchestrator in line 23.

The P2P data contains either cipher data (line 5) or a data reference (line 15).

- In the first case, the requester orchestrator asks to its data signature manager the signature of the received cipher data in line 6. In line 7, it asks the owner orchestrator the symmetric cipher key. The owner orchestrator sends the cipher key or the data (unencrypted) if its data signature differs from the requester's one. Then, the requester orchestrator asks the cipher manager to decrypt the cipher data in line 9.
- In the second case, in line 16, the requester orchestrator asks the owner orchestrator the data.

Finally, the orchestrator returns the requested data to the application (line 18).

Retrieve(key, requester, signature): The owner orchestrator uses the retrieve function to answer the request of the requester. Figure 7 shows the retrieving algorithm. In line 2, the owner orchestrator asks its storage manager to get locally the data corresponding to the received key. Then, in line 3, it asks its policy manager to check if the requester user has the right to access data. If yes, the orchestrator asks the trust manager to find the requester trust level in line 4. If the trust level is higher or equal to the trust level specified in the corresponding PP, in line 8, the owner orchestrator asks the data signature manager to check if the requester data signature differs from the local signature of the data. In this case, the requester has a valid cipher data and the owner sends him the

```
Owner orchestrator
0: OwnerData retrieve(key, requester, signature)
1: begin
2:   localData = storageManager.localGet(key);
3:   if (policyManager.check(localData.PPID,requester))
4:     requesterTrustL = trustManager.searchTrustLevel(
                                                 requester);
5:     privacyPolicy = policyManager.get(
                                      localData.PPID());
6:     if( requesterTrustL >= privacyPolicy.minTrustL )
7:       localSignature = dataSignatureManager.
                    getSignature(localData.cipherData);
8:       if (localSignature.equals(signature)) do
9:          cipherKey = cipherManager.get(
                                       localData.data,
                                        privacyPolicy);
10:         return OwnerData(cipherKey);
11:       endIf;
12:       return OwnerData(localData.data);
13:     endif;
14:   endIf;
15:   return null;
16: end;
```

Fig. 7. Retrieve() function

symmetric cipher key[5] in line 10. If data are damaged, the owner sends to the requester the corresponding data in line 12. If data are damaged, there is a high probability that the server has modified the data. This means that it violates data privacy and this cheating should be punished, for example by lowering the trust level of the server.

In the next we explain briefly the three ways of obtaining trust levels.

d) SearchTrustL(requester): If the owner trust manager has the trust level of the requester in its trust table, this level is returned directly and the owner does not have to contact other peers, otherwise one of the following methods is used.

- *With-friends algorithm.* In this method, each peer has at least one friend that the owner contacts to obtain the requester trust level. Each received trust level (*RTL*) is weighted with the trust level (*FTL*) of the sending friend. The final trust level is computed from the received trust levels. Searching for the requester trust level is recursive. If a friend does not have the requested trust level it asks for it to its friends and the number of nested levels is incremented. Recursion is limited by a predefined number of iterations (MaxDepth). The maximum number of contacted friends can also be limited to a predefined number.
- *Without-friends algorithm.* In this method, each peer does not have friends. The algorithm will proceed in the same way as the with-friends algorithm. However, instead of contacting friends, an owner will contact the peers in its finger table ($O(LogN)$ peers).
- *With-or-without-friends algorithm.* In this method, each peer may have friends or not. In this case, if an owner has some friends, it uses the with-friends algorithm, otherwise it uses the without-friends algorithm.

[5]Maybe the unique completely secure way of transmitting keys is from hand to hand between individuals. When a peer requests data, server peers return the encrypted data and the data owner reference that stores the decryption key. Owner peers use public-key cryptography to send decryption keys to requester peers.

C. Cost analysis

This section presents a cost analysis of PriServ. In particular we present the costs of the *publishing* and *requesting* functions.

1) Publishing cost: By using the DHT, $O(logN)$ messages are needed to publish each key. In PriServ, the number of keys is equal to the number of entries of the private table (ept). Additional costs induced by the cypher key generation and the data encryption are negligible vis-a-vis the network costs. Thus, the publishing cost is:

$$C_{Publish} = \sum_{i=1}^{ept} O(logN) = O(ept * logN)$$

The maximum value of ept is equal to the number of shared data (*nbData*) multiplied by the number of purposes (*nbPurpose*) multiplied by the number of operation (*nbOperation*), i.e., at worst, each data item is shared for all purposes and all operations:

$$CMax_{Publish} = O(nbData{*}nbPurpose{*}nbOperation{*}logN)$$

We can see that the number of purposes and operations affects the publishing cost. Previous studies have shown that considering ten purposes allows to cover a large number of applications [20]. Used with ten purposes (by data item) and three operations PriServ incurs a small overhead. Overall, the publishing cost remains logarithmic.

2) Requesting cost: The requesting cost is the result of three costs: get() cost, retrieving cost, and trust level searching cost. We disregard access control, signature calculation and decryption costs which are negligible vis-a-vis the network costs. The get() cost is in $O(logN)$ and the server returns its answer in one message. For data retrieving, a requester needs one message to contact an owner and vice versa. The trust level searching costs are shown in [14]. To resume, the requesting cost is:

$$\begin{aligned} C_{Requesting} &= C_{Get} + C_{Retrieving} + C_{SearchTrustL} \\ &= O(logN) + 3 + O((max(logN, NF))^{MaxDepth}) \\ &= O((max(logN, NF))^{MaxDepth}) \end{aligned}$$

We can see that the overhead introduced by PriServ comes from trust level searching. We have shown in [14] how this cost can be reduced and stabilized to a minimum cost.

IV. Performance

PriServ is under development. It is being developed in Java. Distribution and requesting functions have been developped and tested. We plan to deploy PriServ and to test it in Grid5000[6], an infrastructure distributed in 9 sites around

[6]www.Grid5000.fr

Simulation parameters		
Variable	**Description**	**default**
n	Number of bits in the key/peer	11
N	Number of peers	2^{11}
FC	Number of friends	2
MaxDepth	Maximum depth of trust searching	11

TABLE IV
TABLE OF PARAMETERS

France, for research in large-scale parallel and distributed systems.

By simulation, this section evaluates the performance in terms of number of messages of the publishing, requesting and trust level searching algorithms.

For the simulation, we use SimJava [11]. We simulate the Chord protocol with some modifications in the $put()$ and $get()$ functions. The parameters of the simulation are shown in Table IV. In our tests, we consider N peers with a number of data keys equal to the number of data multiplied by the number of purposes[7]. Data and peer keys are selected randomly between 0 and 2^n. In our simulation, we set n to 11 which corresponds to 2^{11} peers. This number of users is largely sufficient for collaborative applications like medical research.

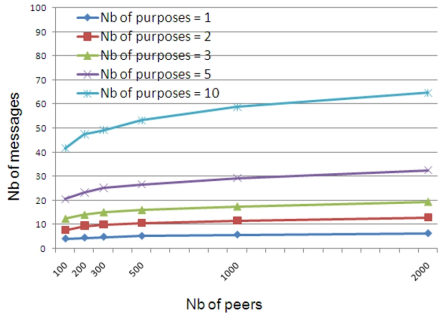


Fig. 8. Publishing cost

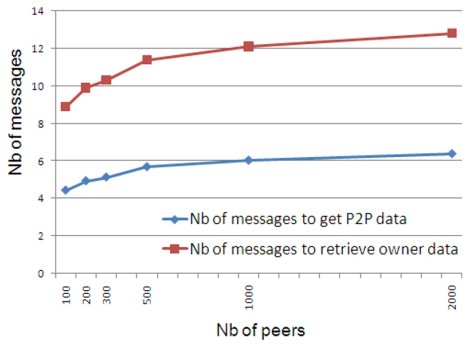


Fig. 9. Requesting cost

A. Publishing Cost

We measure the number of messages for publishing one data item in function of the number of peers. Figure 8 illustrates 5 measures where the number of purposes goes from 1 to 10. We can observe that the distribution cost is logarithmic and increases with the number of purposes. We recall that having 10 purposes for each data item is an extreme case. If the operation parameter of publishing functions is taken into account, this results may be multiplied by 3 (at worst) but the cost remains logarithmic.

B. Requesting Cost

We measure the number of messages for requesting one data item in function of the number of peers. Figure 9 shows two costs. The first cost is the number of messages to get the P2P data. The second cost is the sum of the first cost and the number of messages to retrieve the owner data or cipher key. We observe that the requesting costs are logarithmic as predicted by our cost model (see Section III-C).

C. Trust Level Searching Cost

We measure the trust level searching cost (in number of messages) versus the number of peers for the three algorithms. In the simulation, we optimize the number of messages as follows. Consider the trust level searching as a tree in which the owner is the root and the depth is equal to the maximum depth of searching. In the simulation, peers who are in the same branch will not be contacted twice.

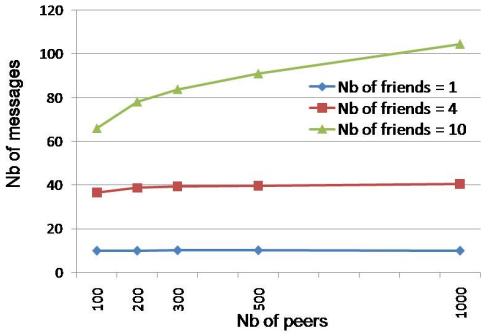
a) With-friends algorithm: We consider that peers have the same number of friends (*NF*) and the maximum depth (*MaxDepth*) is set to 11 (the highest maximum depth we allow in the simulation). Figure 10.a illustrates 3 measures where we consider 1, 4 and 10 friends. Those measures are slightly different of the cost model where the cost is $O(NF^{MaxDepth})$ thanks to our tree-based optimization and because the probability to contact twice a peer in a system of 100 peers is higher than in a system of 1000 peers. That is why in Figure 10.a, the trust level searching cost increases with the number of peers. We observe that for a small number of friends the trust level searching costs depends only on the number of friends as predicted by our cost model.

b) Without-friends algorithm: Figure 10.b illustrates 4 measures where the maximum depth of searching varies between 1, 2, 3 and 11. We recall that the number of contacted peers is $log(N)$. We observe that the trust level searching costs is logarithmic for small values of depth. This cost increases with the maximum depth of searching as predicted by our cost model.

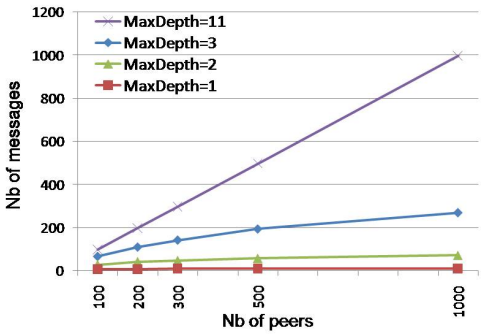
c) With-or-without-friends algorithm.: We consider in our simulation that the probability that a peer has friends is 0.9. Figure 10.c illustrates 3 measures where the maximum depth of searching varies between 5, 8 and 11. We observe that the trust level searching cost is rather logarithmic for small values of depth. This cost increases with the maximum depth as predicted by our cost model[8].

[7]In this simulation the operation parameter of the publishData() and publishReferences() functions is not taken into account but this does not affect our results because currently we only consider 3 operations.

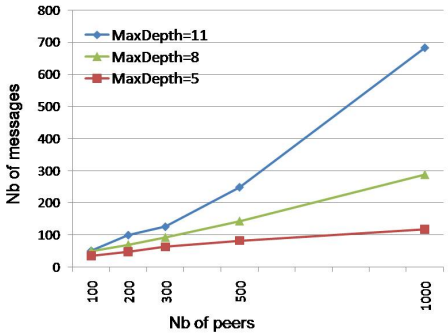
[8]We do not measure the trust level searching cost versus the number of friends which will be the same as the with-friends case.



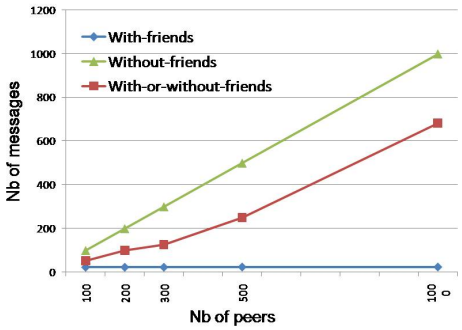
a. With-friends algorithm



b. Without-friends algorithm



c. With-or-without-friends algorithm



d. Comparison of the three algorithms

Fig. 10. Trust level searching costs

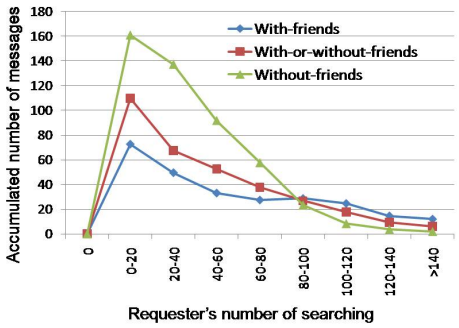


Fig. 11. Stabilization of the trust level searching cost

d) Comparison: Figure 10.d compares the three algorithms seen above. We consider a number of friends equal to 2 and a maximum depth equal to 11. As predicted before, the with-friends case introduces the smallest cost while the without-friends case introduces the highest. However, intuitively, the probability to find the trust level is higher in the without-friends algorithm than in the with-friends algorithm. This is due to the fact that the number of contacted peers is higher in the without-friends algorithm, which increases the probability to find the trust level. We estimate that the with-or-without-friends algorithm is the most optimized because it is a tradeoff between the probability to find the requester trust level and the trust level searching cost.

Stabilization of the Trust Level Searching Cost: We now focus on the number of messages used to search the trust level of a requesting peer versus the number of its requests. Here we consider the three algorithms of trust (see Figure 11). We observe that the number of messages decreases and stabilizes after a number of searches. This is because the more a peer requests for data, the more it gets known by the peers in the system.

When peers ask for a trust level, answers are returned in the requesting order and the trust tables are updated with the missing trust level. Thus, the trust tables evolve with the number of searches. After a while, these tables stabilize. Thus, the number of messages for searching trust levels is reduced to a stable value. This value is not null because of the dynamicity of peers[9].

We also observe in Figure 11, that the trust level searching cost in the without-friends algorithm stabilizes first. This is due to the fact that a larger number of peers are contacted in this algorithm. The with-or-without-friends algorithm comes in second place, and the with-friends algorithm comes last. As we have seen in the comparison of the three algorithms, we find again that the with-or-without-friends algorithm is the most optimized because it is a tradeoff between the time to stabilization and the trust level searching cost.

The trust level searching cost makes requesting cost linear. Nevertheless, we proved that this cost is eventually reduced in systems where the interaction among peers is high.

V. Related Work

Our work is related to three domains, namely, Hippocratic databases, P2P systems and trust.

A. Hippocratic databases

The first work that uses purposes in data access is Hippocratic databases [2]. Inspired by the Hippocratic Oath and

[9]In our simulation, we consider that the number of peers joining the system is equal to those leaving the system. Thus, there are always new peers which do not know the requester trust level.

guided by privacy regulations, authors propose ten principles that should be preserved in Hippocratic databases, namely, purpose specification, consent of the donor, limited collection, limited use, limited disclosure, limited retention, accuracy, safety, openness and compliance. Subsequent works have proposed solutions for Hippocratic databases. In [17], the goal is to enforce privacy policies within existing applications. To do so, authors propose a query modification algorithm that ensures cell-level privacy protection. In [1], authors address the problem of how current relational DBMS can be transformed into their privacy-preserving equivalents. From specifications of privacy policies, they propose an algorithm that defines restrictions (on columns, rows and cells) to limit data access. In [4], authors propose query modification techniques and Role-Based Access Control (RBAC) to ensure data privacy based on purposes. They propose to organize purposes in a tree hierarchy where the root is the most general purpose and the leafs the more specific ones. In this way, if data access is allowed for a purpose x, all descendant purposes of x are also allowed. They also propose data labeling (with allowed purposes) at different granularity levels (table, column, row, cell). In addition, they propose some SQL modifications to include purposes, for instance **Select** column-name **From** table-name **For** purpose-name.

In this paper, we use the Hippocratic database principles [2], mainly, access purposes. As in [17] we propose enforcing privacy policies without modifying existing database applications. We propose to extend P2P functionalities with a service that uses privacy policies tables where data and purposes are linked. Similar to [4], in PriServ the requesting() function (i.e., query) includes the purpose name. Unlike [4] relational tables are not modified with labeling.

B. Privacy in P2P systems

OceanStore [16] is a utility infrastructure designed for global scale persistent storage which relies on Tapestry [27]. It was designed to provide secure highly available access to persistent objects. In OceanStore, non public data is encrypted and access control is based on two types of restrictions: *reader* and *writer* restrictions. In the *reader* restriction, to prevent unauthorized reads, data are encrypted (symmetric-key cryptography) and encryption keys are distributed to users with read permissions. To revoke the read permission, the data owner requests that the replicas be deleted or re-encrypted with a new key. A malicious reader is able to read old data from cached copies or from misbehaving servers that fail to delete or re-key. This problem is not specific to OceanStore, even in conventional systems there is no way to force a reader to forget what has been read.

To prevent unauthorized writes, writes must be signed so that well-behaved servers and clients can verify them against an access control list (ACL). The owner of an object can choose the ACL for an object by providing a signed certificate. ACL are publicly readable so that server peers can check whether a write is allowed. Thus, servers restrict writes by ignoring unauthorized updates. As in PriServ, OceanStore uses ACL to control data access (write operation) but contrary to PriServ, those ACL are public. In PriServ, ACL (that are attached to published data or references) are sent only to corresponding data servers.

PAST [21] is a P2P file storage system that relies on Pastry [22] to provide strong persistence and high availability. In PAST, requester peers trust owner and server peers thanks to a smartcard hold by each node which wants to publish data in the system. A private/public key pair is associated with each card. Each smartcard's public key is signed with the smartcard issuer's private key for certification purposes. The smartcards generate and verify various certificates used during insert and reclaim operations and they maintain secure storage quota system. A smartcard provides the node ID for an associated PAST node. The node ID is based on a cryptographic hash of the smartcard's public key. The smartcard of a user wishing to insert a file into PAST issues a file certificate. The certificate contains, among others, a cryptographic hash of the file's contents (computed by the requested node) and the fileId (computed by the smartcard). PAST as PriServ attempts to be resilient to malicious servers but contrary to PriServ, PAST do not trust in data owners.

Freenet [6] is a distributed storage system which focuses on privacy and security issues. The underlying P2P network is loosely structured because the policies it employs to determine the network topology and data placement are not deterministic. Freenet uses anonymous communications. Messages are not send directly from sender to recipient, thus, uploader and downloader anonymity is preserved. Besides, all stored files are encrypted so that a node can deny the knowledge of the content. PriServ allows data privacy protection and does not address peer anonymity.

Several P2P systems propose access control services. It is important to notice that only two of them rely on DHT systems (OceanStore and PAST). Similar to PriServ, they consider servers as untrusted and propose techniques to limit their malicious acts. Like OceanStore and Past, PriServ uses encryption to protect data (only when publishing data content in the system not data references). In those systems, the lack of authentication is overcome by the distribution of the necessary keys for accessing data content to a subset of privileged users.

Data encryption also allows servers to deny the knowledge of the content they store and protects them if data stored in the system is illegal. Nevertheless, encryption is insufficient to protect data privacy because once data are decrypted, they can be disclosed and used for different purposes violating data owner's preferences. In this aspect PriServ goes beyond cryptography techniques by basing data access also on purposes and operations. To obtain data, requesters should specify the purpose (analysis, experiments, marketing, etc.) and the operation (read, write, disclosure) for which data are requested. This explicit request commits clients to use data only for specified purposes and operations. Legally, this compromise, may be used against malicious requesters if it is turned out requested data have been used for purposes/operations non expressed in data requests.

C. Trust management

Other works propose trust-based management systems [15], [18], [24]. In [24], a considerable number of trust models and algorithms have been presented to tackle the problem of decentralized trust management.

EigenTrust [15] is based on the notion of transitive trust. If a peer A trusts a peer B, A also trusts peers trusted by B. All peers calculate a local trust level for each peer based on the satisfactory or unsatisfactory transactions that they have with it. If a peer B needs to verify trustworthiness of a peer C, B collects statistics about the behavior of peer C and computes a global trust rating for C. EigenTrust employs a shared global history of peer interactions to identify potential malicious providers. We consider that global history schemes are complicated, they require long periods of time to collect statistics and to compute global ratings. Priserv does not use a global history, a peer does not have to contact all peers in order to search for a trust level. Thus PriServ require less time and messages, and have a lower searching trust cost.

[18] presents a P2P resource-sharing network in presence of malicious peers, which uses only limited sharing between peers. It presents a voting-based reputation system that significantly mitigates the deleterious effects of malicious peers, by sharing information with a small group of peers. In [18], a peer A asks for a trust level to a limited number of peers called friends. A friend is a trustworthy peer from the point of view of A. A calculates a mean value from collected trust levels which are weighted by the trust levels of friends. PriServ uses [18] in order to search for trust levels. Unlike [18] where the trustworthiness of data providers is verified, in PriServ the trustworthiness of data requesters is verified.

VI. Conclusion

This paper proposes an approach to prevent data privacy violation based on data owner's preferences on P2P data sharing applications. A P2P privacy model and its implementation, named PriServ, is presented. The originality of our approach is a data access control that uses *access purposes* and *operations* in P2P systems. During access control, optionally, reputation techniques are used to verify requesters's trustworthy. Implementation relies on DHT-based P2P systems by using the basic put and get functions. Our cost analysis reveal that publishing data in the system conserves the logarithmic cost of the traditional put function. Concerning requesting, by using peer's reputation, costs are linear, otherwise costs are logarithmic like in the traditional get function.

References

[1] Rakesh Agrawal, Paul Bird, Tyrone Grandison, Jerry Kiernan, Scott Logan, and Walid Rjaibi. Extending Relational Database Systems to Automatically Enforce Privacy Policies. In *IEEE Conference on Data Engineering (ICDE)*, 2005.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic Databases. In *Very Large Databases (VLDB)*, 2002.

[3] Reza Akbarinia, Vidal Martins, Esther Pacitti, and Patrick Valduriez. Design and Implementation of APPA. *Global Data Management (Eds. R. Baldoni, G. Cortese, F. Davide), IOS Press*, 2006.

[4] Ji-Won Byun, Elisa Bertino, and Ninghui Li. Purpose Based Access Control of Complex Data for Privacy Protection. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2005.

[5] Ji-Won Byun and Ninghui Li. Purpose Based Access Control for Privacy Protection in Relational Database Systems. *The VLDB Journal*, 17(4), 2008.

[6] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1), 2002.

[7] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. The Platform for Privacy Preferences 1.0. (P3P1.0) Specification. W3C Recommendation, April 2002.

[8] Frank Dabek, Ben Y. Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *International Peer To Peer Systems Workshop (IPTPS)*, 2003.

[9] FIPS. 180-1 Secure Hash Standard. U.S. Department of Commerce/NIST. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield VA, 1995.

[10] Pedro Furtado. Schemas and Queries over P2P. In *Database and Expert Systems Applications (DEXA)*, 2005.

[11] Fred Howell and Ross McNab. Simjava: a Discrete Event Simulation Library for Java. In *Society for Computer Simulation (SCS)*, 1998.

[12] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Very Large Databases (VLDB)*, 2003.

[13] Mohamed Jawad, Patricia Serrano-Alvarado, and Patrick Valduriez. Design of PriServ, A Privacy Service for DHTs. In *International Workshop on Privacy and Anonymity in the Information Society (PAIS), collocated with EDBT*, 2008.

[14] Mohamed Jawad, Patricia Serrano-Alvarado, and Patrick Valduriez. Protecting Data Privacy in Structured P2P Networks. In *International Conference on Data Management in Grid and P2P Systems (Globe)*, 2009.

[15] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P networks. In *ACM World Wide Web Conference (WWW)*, 2003.

[16] John Kubiatowicz, David Bindel, Yan Chen, Steven E. Czerwinski, Patrick R. Eaton, Dennis Geels, Ramakrishna Gummadi, Sean C. Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Y. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[17] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David J. DeWitt. Limiting Disclosure in Hippocratic Databases. In *Very Large Databases (VLDB)*, 2004.

[18] Sergio Marti and Hector Garcia-Molina. Limited Reputation Sharing in P2P Systems. In *ACM Conference on Electronic Commerce (EC)*, 2004.

[19] Sergio Marti and Hector Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. *Computer Networks*, 50(4), 2006.

[20] 1.0 P3P Purposes of Data Collection Elements. http://p3pwriter.com/LRN_041.asp.

[21] Antony Rowstron and George House. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In *Symposium on Operating Systems Principles (SOSP)*, 2001.

[22] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *ACM/IFIP/USENIX Middleware Conference (MIDDLEWARE)*, 2001.

[23] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2001.

[24] Girish Suryanarayana and Richard N. Taylor. A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. Technical report, UCI Institute for Software Research, university of California, Irvine, 2004.

[25] Igor Tatarinov, Zachary G. Ives, Jayant Madhavan, Alon Y. Halevy, Dan Suciu, Nilesh N. Dalvi, Xin Dong, Yana Kadiyska, Gerome Miklau, and Peter Mork. The Piazza Peer Data Management Project. *ACM SIGMOD Record*, 32(3), 2003.

[26] Alan F. Westin. Privacy and Freedom. In *Atheneum*, New York, 1967.

[27] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.