Distributed and Secure Access Control in P2P Databases

Angela Bonifati[1], Ruilin Liu[2], and Hui (Wendy) Wang[2]

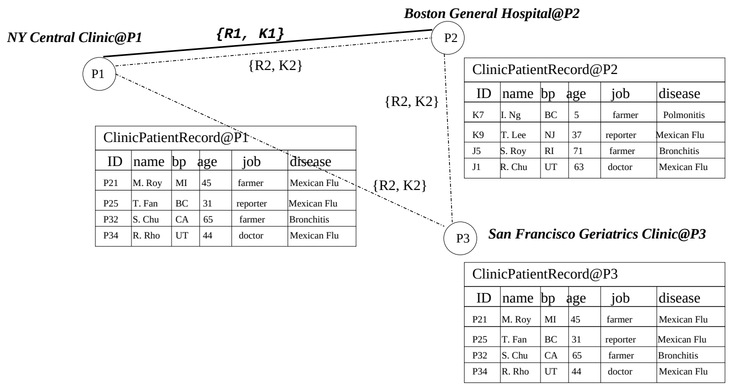
[1] Italian National Research Council (CNR)
Via P. Bucci 41C, I-87036 Rende, Italy
bonifati@icar.cnr.it
[2] Stevens Institute of Technology
Dept. of CS, Castle Point on Hudson, Hoboken, NJ, USA, 07030
rliu3@cs.stevens.edu, hwang@cs.stevens.edu

Abstract. The intent of peer data management systems (PDMS) is to share as much data as possible. However, in many applications leveraging sensitive data, users demand adequate mechanisms to restrict the access to authorized parties. In this paper, we study a distributed access control model, where data items are stored, queried and authenticated in a totally decentralized fashion. Our contribution focuses on the design of a comprehensive framework for access control enforcement in PDMS sharing secure data, which blends policy rules defined in a declarative language with distributed key management schemes. The data owner peer decides *which data to share* and *whom to share with* by means of such policies, with the data encrypted accordingly. To defend against malicious attackers who can compromise the peers, the decryption keys are decomposed into pieces scattered amongst peers. We discuss the details of how to adapt distributed encryption schemes to PDMS to enforce robust and resilient access control, and demonstrate the efficiency and scalability of our approach by means of an extensive experimental study.

1 Introduction

Peer Data Management Systems (P2P databases or PDMS in short) introduce a revolutionary paradigm for distributed data management [1], [2], [3]. They provide fully decentralized and extensible data management architecture. In ordinary PDMS, data is freely shared in the network and peers unconditionally trust the other participants. However, since the data may contain sensitive information, flexible and effective access control on such data becomes crucial. A number of proposals consider the problem of enforcing access control in P2P networks [4–7]. They focus on the design of the architecture [4], the persistent storage [5], distributed file systems [6] and administrative distribution [7]. However, none of them considers flexible access control mechanisms that can effectively support multiple access policies as well as efficient and secure query access to PDMS.

In this paper, we design a robust distributed access control mechanism for large-scale PDMS. In particular, each peer is allowed to specify the access control requirements on its local data by means of policy rules; the other peers will


NY Central Clinic@P1
{R1, K1}
Boston General Hospital@P2
P1
P2
{R2, K2}
{R2, K2}
{R2, K2}
P3
San Francisco Geriatrics Clinic@P3

ClinicPatientRecord@P1

ID	name	bp	age	job	disease
P21	M. Roy	MI	45	farmer	Mexican Flu
P25	T. Fan	BC	31	reporter	Mexican Flu
P32	S. Chu	CA	65	farmer	Bronchitis
P34	R. Rho	UT	44	doctor	Mexican Flu

ClinicPatientRecord@P2

ID	name	bp	age	job	disease
K7	I. Ng	BC	5	farmer	Polmonitis
K9	T. Lee	NJ	37	reporter	Mexican Flu
J5	S. Roy	RI	71	farmer	Bronchitis
J1	R. Chu	UT	63	doctor	Mexican Flu

ClinicPatientRecord@P3

ID	name	bp	age	job	disease
P21	M. Roy	MI	45	farmer	Mexican Flu
P25	T. Fan	BC	31	reporter	Mexican Flu
P32	S. Chu	CA	65	farmer	Bronchitis
P34	R. Rho	UT	44	doctor	Mexican Flu

Fig. 1. Sharing Sensitive Information in a PDMS.

attempt to retrieve its data by asking queries; however, only the peers who have appropriate data access can get the answer, if any. Due to the fact that the network lacks a centralized authentication module, one viable approach to enforce access control is by using encryption: the sensitive data within the peer databases is encrypted; only accessible peers can obtain appropriate keys to decrypt and access the data. Although it is an effective and popular approach in several contexts [8–10], adapting it to PDMS poses a few challenges.
Challenge #1: In our threat model, the malicious attackers can compromise any peer in the network and thus can learn all information, including the decryption keys, held by the compromised peers. It is straightforward that simply storing decryption keys in the network cannot effectively protect the data access. Thus, the first challenge is how to guard the decryption keys against malicious attackers that may enter the network.
Challenge #2: There may exist multiple access control policy rules that involve the same set of data values. Careless design of the encryption scheme, including the granularity and decryption keys, to enforce these rules, will lead to expensive overhead for data storage, communication, and query evaluation. For a better illustration of this challenge, we use the following example.
Example 1. Figure 1 illustrates a PDMS designed for the Health Care including hospitals, clinics, and research labs. For simplicity, assume that each peer holds a relational table `ClinicPatientRecord`, with the information about the patients' name, birthplace, age, job and disease. Assume one of the peers, the NY central clinic at peer P_1, denoted as `NYCentralClinic@P1`, has two sets of access control requirements. First, to help conduct the research on the 'Swine Flu' disease, it would like to share the *name*, *birthplace*, and *age* information of its patients who got 'Swine Flu' with the hospitals in the Eastern Coast of the U.S. (i.e. P_2). Second, for a different study of the correlation of birthplaces, jobs and diseases, it is willing to share the *birthplace*, *job*, and *disease* information of its patients with all other peers in the network (i.e. P_2 and P_3).

Assume that the enforcement of the first access control rule (*R1*), denoted in Figure 1 as a solid line, will result in that the `NYCentralClinic`, the owner of the

`ClinicPatientRecord@P1` database, encrypts the *name*, *birthplace*, and *age* data and share the decryption keys with `BostonGeneralHospital@P2`. Furthermore, the enforcement of the second policy rule (`R2`) will lead to the encryption of the *birthplace*, *job*, and *disease* data, with the decryption keys shared with both `BostonGeneralHospital@P2` and `SanFrancescoClinic@P3` (highlighted with dashed lines in Figure 1). As the *birthplace* values for the same sets of tuples describing all patients are covered by both rules, if these two rules use different decryption keys, these *birthplace* values need to be replicated for separate encryptions by different peers, which may incur both data overhead and expensive network communication. Thus, a careful design of the encryption scheme is needed in order to identify the common values encrypted by multiple access control rules and thus overcome the above problem. □

Challenge #3: In PDMS, network updates occur very frequently. Leave/join of peers can lead to the updates of access control configuration. For instance, new access control requirements may be introduced for the newly inserted peers. Efficient mechanisms that enforce the updates of access control on PDMS are vital to the security and performance of the system.

In this paper, we propose a comprehensive framework for access control enforcement in PDMS, using encryption, that provides the following important capabilities:

(i) Robust access guard against malicious attackers that can compromise all information of any peer in the network,

(ii) Fully decentralized authentication, as the network lacks centralized administration, and

(iii) Resilience to updates on the network as well as the access control policy.

We adopt a cryptographic approach that provides robust, decentralized, and resilient access control on PDMS. Our contributions include:

(1) We define a declarative distributed query language to specify access control policy in PDMS. Such language is based on SQL, yet powerful enough to allow expressive policies for access on various granularities.

(2) We enforce the access control policy rules by encryption; only authorized users can obtain the decryption key and consequently gain the access to the data. To support decentralized and resilient management of decryption keys, we adapt a classic cryptographic secret sharing protocol called (m, n) threshold scheme [11] to PDMS. In particular, a decryption key K is split into n pieces; only by collecting at least $m < n$ key pieces the peers can reconstruct K.

(3) We address the challenges of key management when multiple access control rules overlap the access on the same data values. These common data values are encrypted by using the same key. We show the relationship of this problem with query containment, restrict to the polynomial case [12], and identify a monotone property for key shares on common values. The monotone property can significantly reduce the number of keys needed for the enforcement of access control rules.

(4) We further investigate how to manage keys when there exist updates on access control policies. We propose an effective scheme that preserves the monotone property of key shares when there are updates on access control policies.

(5) We demonstrate the efficiency and scalability of our approach by means of a comprehensive experimental study.

To the best of our knowledge, ours is the first attempt to address the challenges of enforcing SQL-based distributed access control policies on *dynamic* PDMS and efficiently handling *updates* on such policies.

The paper is organized as follows. Section 2 explains the model setup, including the access control framework and enforcement mechanism. Section 3 discusses the key management when there are multiple access control rules accessing the same data as well as updates on the rules. Section 4 presents our experimental study and Section 5 discusses the related work. Finally, Section 6 concludes the paper.

2 System Model

2.1 Network Model

A P2P network is an ad-hoc collection of peers willing to share their data. Each peer contributes its local storage to the global data store. Each peer is equipped with a relational database. In this paper, we assume that every peer in the network has the same database schema as the rest of the network; the extension to multiple heterogeneous schemas is orthogonal to the proposed techniques.

2.2 Threat Model

We assume every peer in the network can be compromised by the attacker. The attacker's intent is to access the data that he/she is not authorized to. He/she can learn all information held by the compromised peers, including the data and decryption keys, and can eavesdrop on the communication among all peers. However, he/she is assumed to be computationally bounded and thus cannot break the underlying cryptographic schemes without knowing the appropriate cryptographic keys.

2.3 Access Control Model

Access Control (AC) Policy. To specify which data is accessible to which peers, we define a declarative access control language. The access control policy is in the format of SQL queries with a *Peer* clause. In particular, an *access control* (AC) rule is in the form of:

```
SELECT target_List
FROM table_List
WHERE WhereExpr
PEER (peer_List | SELECT (peer_ID | *) FROM peer_List WHERE P_WhereExpr),
```

where *target_List* is the list of target attributes $A_1, \cdots, A_m$ in the AC rule, *table_List* is the list of relations $T_1, \cdots, T_k$, and *WhereExpr* is an arbitrary conjunctive predicate on *table_List*. The PEER clause may contain the *peer_List* as a list of peers $P_1, \cdots, P_n$ that have access to the data specified by the AC rules. In alternative, the PEER clause is expressed by means of an arbitrary query on the relation *peer_List*, with an arbitrary conjunctive predicate *P_WhereExpr* on *peer_List*. We use R^Q to denote the part of AC rule R without the PEER clause.

Example 2. The access control rules R_1 and R_2 below specify the access control requirement in Example 1:

```
R1: SELECT name, birthplace, age
      FROM ClinicPatientRecord@P1
      WHERE disease = ‘Swine Flu’
      PEER (SELECT peer_ID FROM peer_List
         WHERE PeerLocation = ‘United States East Coast’)
R2: SELECT birthplace, job, disease
      FROM ClinicPatientRecord@P1
      PEER (SELECT * FROM peer_List)
```

We assume that all peers have access to all data *by default*. If a peer wants to limit the access to its own data, it defines one or more AC rules on its local peer database D; peers are not allowed to specify AC rules on data on other peers.

Overlap, Containment, and Equivalence of AC Rules. In the following, we use $\mathcal{R}$ and R to specify a set of access control rules and a specific access control rule. Let D be a database and R be a specific AC rule, we use $R^Q(D)$ to denote the set of tuples as result of evaluating R^Q on D. As customary, we assume that each tuple $t = \langle v_1, v_2, \cdots, v_n \rangle$ of the database D has values v_i, $1 \leq i \leq n$, where each v_i is an element of $dom(A_i)$, A_i being the name of an attribute in a relational schema. Moreover, each tuple t has a unique ID, given by its primary key. We denote the ID value of the tuple t as $t(ID)$, and the value v_i of the tuple t for attribute A_i as $t(A_i)$. Given two tuples t and t', we say that t and t' intersect ($t \cap t'$) if it exists at least one attribute A_i such that the values are the same, i.e. $t(A_i) = t'(A_i)$.

Definition 1. *Given a database D and two AC rules R_i and R_j on D, we define $R_i(D) \cap R_j(D) = \{t | t \in R_i^Q(D), \exists t' \in R_j^Q(D), t \cap t', t(ID) = t'(ID)\}$, and $R_i(D) - R_j(D) = R_i(D) - R_i(D) \cap R_j(D)$.*

As stated in Definition 1, the overlapping tuples of R_i and R_j *are exactly the same tuples*. In our context, AC rules are defined on the same peer database, thus the returned tuples have exactly the same identifier. We then define containment and equivalence of AC rules R_i and R_j. While the classical definition of query containment and equivalence [12] require that evaluating R_i and R_j on the database D returns compatible tuples (i.e. having the same schema), the definition below relaxes such requirement by identifying compatible tuples as those tuples having at least one attribute A_i in common, and the same ID value.

Definition 2. *Given two AC rules R_i and R_j, we say R_i* overlaps *R_j, denoted as $R_i \cap_{ac} R_j$ if $R_i(D) \cap R_j(D) \neq \oslash$. We say R_i* is contained in *$R_i(D)$, denoted as $R_i(D) \subseteq_{ac} R_j(D)$, if $\forall$ value $v \in R_i(D)$, $\exists v' \in R_i(D) \cap R_j(D)$ s.t. $v = v'$ and $v(ID) = v'(ID)$. We say the AC rule R_i* is equivalent to *the rule R_j, denoted as $R_i =_{ac} R_j$, if $R_j \subseteq_{ac} R_i$ and $R_i \subseteq_{ac} R_j$.*

Checking query containment in general is well-known to be NP-complete [13]. As shown by Saraiya [14] and Chekuri et al. [12], conjunctive query containment of acyclic queries can be solved in polynomial time. Since we only consider AC rules R whose R^Q are acyclic conjunctive queries, checking the containment of AC rules has polynomial time complexity, even though they may return the instances of different schemas. We have:

Lemma 1. *Given two AC rules R_i and R_j, checking whether $R_i \subseteq_{ac} R_j$ has polynomial time complexity.*

Cryptography-based Access Control Enforcement. P2P networks are characterized by the complete lack of centralized and trusted components, which brings difficulty to the design of access control mechanism. In view of this, we rely exclusively on cryptography to enforce the access control policy and provide access control to PDMS. In particular, each access control rule R corresponds to a set of encryption blocks; only peers who have access to the data in the encryption blocks can possess the corresponding decryption keys. The reason of using encryption is that cryptographic operations (such as keys) and authentication can be distributed among several peers.

To implement the cryptography-based access control mechanisms, we adopt a pioneering secret sharing protocol, namely the (m, n) threshold scheme [11], exhibiting an $O(n\ log^2(n))$ complexity, and considered fast enough for practical key management schemes. This secret sharing protocol is ideally suited to applications in which a group of mutually suspicious individuals must cooperate for a common goal. It is useful in distributed scenarios where secrecy and integrity of information needs to be protected, and make particular sense in a PDMS. Informally, the (m, n) threshold scheme [11] distributes a secret by a *dealer* to n *participants*, each of which is allocated a share of the secret. The secret can only be reconstructed when $m < n$ shares are combined together; individual shares are of no use on their own.

We adapt the above secret sharing protocol to PDMS, by considering every single decryption key as a secret. Every peer can be a participant. The dealer is the data owner who distributes the key pieces to the peers that he/she grants the access. In this way, discretionary access control (DAC) [15] is supported. We assume the *dealer* peers are *transiently honest*, i.e., they are considered honest when they split the decryption keys into key pieces, and destroy the decryption keys after the key pieces have been distributed. When there comes the need for data decryption, every peer, including the data owner, has to collect other key pieces to reconstruct the keys. This scheme supports fully decentralized authentication (as no single entity needs to be fully trusted), robust access control (as compromising any single peer will not enable the attacker to decrypt the data), and resilience of the system (as some of the peers may not be available or decide not to share their pieces of decryption key with other peers) to a number of up to $n - m$ simultaneously leaving/failing or compromised peers. Furthermore, this technique provides adjustable degree of attack protection and fault tolerance by controlling the value of m.

Encryption Granularity. Our access control policy allows access control specification at various granularities, including the individual values, attributes, tuples, and the whole database, as long as these are the output by the AC rules. The granularity at which data objects are encrypted is closely tied to the efficiency of handling decryption keys and processing queries on the decrypted data. There is a whole space of options here, leading to the observation that a finer encryption granularity (i.e. value-level or tuple-level granularity) would lead to excessive overhead, due to an unmanageable high number of keys in the network. By opposite, a database-level granularity would restrict the capability of sharing smaller data fragments in realistic distributed scenarios. We thus opted for a practical hybrid solution that associates a decryption key with the set of tuples covered by each AC rule. We denote such a set of tuples as an *encrypted block*. This rule-based encryption mechanism supports flexible encryption granularity that is decided by the AC rules. In particular, the enforcement of encryption will result in *one or multiple encrypted blocks*; any rule that does not overlap with other rules is enforced in the form of an individual encryption block, while the rules that overlap with the others lead to multiple blocks. Details of the construction of encryption blocks and decryption keys are given in Section 3.

Node Authentication. In our framework, only authorized nodes can collect the key pieces and reconstruct the key. We adopt a certificate-based approach, in which certification services, such as certificate issuing, renewal and revocation, are distributed by using threshold sharing of the certificate signing key. Our node authentication procedure is inspired by previous work on ad-hoc mobile networks [16, 17]. Due to the lack of space, we do not discuss it further.

3 Key Maintenance with Multiple Rules and Updates

In this section, we first discuss how to encrypt the database when there exist overlapping AC rules (Section 3.1). Then, we investigate how to enforce the access control when there are updates on the network, which especially causes the updates of AC rules (Section 3.2).

3.1 Overlapping AC Rules

There may exist tuples that are accessible by multiple peers specified via different access control rules. For instance, the AC rules R_1 and R_2 in Example 2 overlap on *birthplace* data. One possible approach is to allow multiple keys, each corresponding to an AC rule, on the shared data. To support such mechanism without making chaos on access control (as Example 1 illustrates), a naive approach is to make as many replicas of the shared tuples as the number of overlapping AC rules, so that each replica is encrypted and accordingly decrypted by a unique key (according to one specific AC rule). However, such approach will introduce both key and data replication overhead and expensive network communication. In order to avoid such overhead, we devise a *blocking-based encryption* mechanism. The basic idea is that the overlapping AC rules will encrypt their shared data as a single block, and always use the same decryption key for that block. We formally define the mechanism as follows. We use R^{PL} to denote the peer list that is specified by the $PEER$ clause of the AC rule R.

Definition 3. *For any pair of AC rules $\langle R_i, R_j \rangle$ such that $R_i \cap_{ac} R_j$:*
(1) $R_i(D) \cap R_j(D)$ is encrypted as a block B_{ij}. Each peer in R_i^{PL} and R_j^{PL} will have a piece of the decryption key K_{ij} of B_{ij}, and
(2) $R_i(D) - R_j(D)$ and $R_j(D) - R_i(D)$ are encrypted as two separate blocks B_i and B_j. Each peer in R_i^{PL} (R_j^{PL}, resp.) is assigned a key piece K_i (K_j, resp.) for decrypting B_i (B_j, resp.).
We say the AC rule R_i (R_j, resp.) is enforced by the encryption blocks B_{ij} and B_i (B_{ij} and B_j, resp.), and the keys needed for the enforcement of R_i (R_j, resp.) are K_{ij} and K_i (K_{ij} and K_j, resp.).

Intuitively, any rule that does not overlap with other rules is enforced in the form of an individual encryption block, while the rules that do overlap with the others lead to multiple blocks. In the remainder, we refer the keys needed for the enforcement of rule R as the *keys of R*.

We observe that the containment of AC rules naturally lead to the containment of keys of these rules, which is stated as following.

Property 1 (Monotonicity). Given two AC rules R_i and R_i such that $R_j \subseteq_{ac} R_i$, let K_i and K_j be the set of keys of R_i and R_j. Then $K_j \subseteq K_i$.

As an extension, if rule R_i is equivalent to rule R_j, the sets of keys K_i and K_j are the same. We call them *equivalent sets of keys*, if $K_i \subseteq K_j$ and $K_j \subseteq K_i$. Clearly, the monotonicity among keys can be used to reduce the number of the keys. We will show in Section 3.2 how to assign the keys while preserving the monotone property, especially in the presence of updates on access control rules.

3.2 Join/Leave of Nodes

Due to network churn, the peers may join and leave the network at will. This behavior may affect the AC rules as follows. First, join/leave of peers will not incur updates on AC rules, then such peers cause updates on AC rules and key pieces that need to be maintained. We discuss both cases in this section. We recall that we do not allow replication of key pieces to not compromise the security of the secret sharing protocol. By contrary, we allow key piece regeneration in order to avoid the key pieces exposed by security breaches and thus enhance security [11]. We discuss two possible scenarios, that the network updates lead to no updates on AC rules, and they do.

No Updates on AC Rules. When a new peer joins the network and is allowed by a data owner to share its data with the existing n peers by using some existing AC rules, the decryption keys are reconstructed by one of the existing peers, and the key pieces are regenerated in order to include the new peer. As shown in [11], since regenerating the key pieces is simply generating a new polynomial function, the cost is affordable. Furthermore, such key regeneration will actually enhance the security of the network, since the key pieces exposed to security breaches are replaced by new values. In Section 4, we study the effect of such regeneration on the network performance.

When a peer leaves, if the peer does not have any key piece, its disconnection does not affect the secret sharing protocol. If the leaving peer p has a key piece, it informs the data owner peer of its leave. The data owner peer checks the number of available key pieces (including the one that p holds). If the number of such key pieces is greater than m, the data owner informs the leaving peer to leave with no action; otherwise, if the number is equal to m, the data owner peer initiates the key reconstruction procedure. Notice that a set of peers may be leaving the network at the same time. In such a case, the above procedure is repeated if the number of participants left minus the number of such peers is equal or less than m. Finally, if the leaving peer is the data owner, it informs all peers that have the key pieces to destroy them.

Thus, deletion of existing key pieces, and leaves of peers would in the worst case lead to periodical refreshment on the key piece by reconstructing the secret and re-sharing it amongst the participants.

Updates on AC Rules. When the new peers join the network, existing peers may specify the access rights of their data to these peers. This may introduce new AC rules. Similarly, leave of the nodes will result in deletion of old AC rules. Inserting/deleting AC rules will introduce additional complexity on the key management. In what follows, we discuss various types of updates and the corresponding key management strategy.

Deletion of old AC rules. Assume the peer P deletes its AC rule R. Then P collects the key shares, re-constructs the key, and decrypts $R(D)$.

Insertion of new AC rules. Let $\mathcal{R}$ be the current set of AC rules on peer P. Assume the join of new peers requires that P defines a new AC rule R_{new}. There are five cases:

Case 1: there exists at least a rule $R \in \mathcal{R}$ s.t. $R \subseteq_{ac} R_{new}$. Let $R_1 \in \mathcal{R}$ be a rule that is maximally contained in R_{new}, i.e., $R_1 \subseteq_{ac} R_{new}$ and there is no other rule $R' \in \mathcal{R}$ s.t. $R_1 \subseteq_{ac} R'$. Let K_1 be the key of R_1. Then peer P encrypts $R_{new}(D) - R_1(D)$ as a block. Let K be the new key for decryption of this block. P distributes the key shares of both K and K_1 to the peers in R_{new}^{PL}.

Case 2: there exist the rules $R_1, \ldots, R_t \in \mathcal{R}$ such that $R_{new} \subseteq_{ac} R_1 \cdots \subseteq_{ac} R_t$. Peer P sorts them as $R_1, \ldots, R_t$. Then first, peer P reconstructs the decryption key of R_t, and decrypts $R_t(D)$. Note that, by this decryption, $R_{t-1}(D)$ will also be decrypted, and so on and so forth, until $R_1(D)$. Second,

– P encrypts $R_{new}(D)$ as a block, and distributes the key shares of the decryption key K to the peers in R_{new}^{PL}.
– For the i-th ($1 \le i \le t$) AC rule in the sorted list, if $i = 1$, P encrypts $R_1(D) - R_{new}(D)$ as a block, and distributes the key shares of K and K_1 to the peers in R_1^{PL}, where K_1 is a new key that decrypts $R_1(D) - R_{new}(D)$; otherwise, P encrypts $R_i(D) - R_{i-1}(D)$ as a block, and distributes the key shares of K_{i-1} and K_i to the peers in R_i^{PL}, where K_{i-1} is the key that decrypts $R_{i-1}(D)$ and K_i is a new key that decrypts $R_i(D) - R_{i-1}(D)$.

Case 3: there exists a rule $R \in \mathcal{R}$ s.t. $R_{new} =_{ac} R$. For this case, both R and R_{new} use the same keys.

Case 4: there exists a rule $R \in \mathcal{R}$ s.t. both Case 1 and 2 fail but $R(D) \cap_{ac} R_{new}(D)$. Then first, peer P reconstructs the key of R and uses it to decrypt $R(D)$. Second, P encrypts $R(D) \cap R_{new}(D)$ as a block, and distributes the key shares of the decryption key K to the peers in both R^{PL} and R_{new}^{PL}. Third, P encrypts $R(D) - R_{new}(D)$ and $R_{new}(D) - R(D)$ as two blocks, and distributes the key shares of K_1 to the peers in R^{PL}, and the key shares of K_2 to the peers in R_{new}^{PL}, where K_1 and K_2 are two new keys for decrypting $R(D) - R_{new}(D)$ and $R_{new}(D) - R(D)$.

Case 5: there does not exist any rule in $\mathcal{R}$ that satisfies any case above. Then P encrypts $R_{new}(D)$ as a block, and distributes the key shares of the decryption key K to the peers in R_{new}^{PL}.

These five cases may all apply to R_{new}. For instance, there may exist the rules R_1, R_2 that meet Case 1 and 2, as well as R_3 that meets Case 3, R_4 that meets Case 4 and R_5 that meets Case 5. Applying the five cases in a random order may result in wrong key assignment. For instance, in the above example, applying Case 2 before Case 1 will ruin the monotone property of the keys between R_1, R_2, and R_{new}. The failure to preserve the monotone property may result in incorrect encryption and thus inappropriate access control enforcement. Therefore, we propose the following construction procedure to assign keys to R_{new}, so that the monotone property of the keys is well preserved for contained AC rules. Initially the candidate rules is $\mathcal{R}$, the whole set of AC rules.
Superset rules (Step 1): For the rules such that $R_{new} \subseteq_{ac} R_1 \cdots \subseteq_{ac} R_m(D)$, sort them in their containment order, starting from R_{new}. Let S_1 be the sorted result, and $\mathcal{R}'$ be the result of $\mathcal{R}$ - S_1.
Intersected rules (Step 2): for $k = n, n-1, n-2, \ldots, 2$, where n is the total number of AC rules in $\mathcal{R}'$, repeatedly check the intersection of k rules $R_1(D) \cap R_2(D) \cap \ldots R_k(D) \cap R_{new}(D)$ that is not empty. For the ith $(1 \leq i \leq n-1)$ step in the loop, only the intersected rules that are not subsets of any that has been recorded in the previous steps are checked. For instance, $R_1 \cap R_2$ is not considered if $R_1 \cap R_2 \cap R_3 \neq \oslash$. Note that the rules that intersect with R_{new} cover the rules that are contained in R_{new}.
Subset rules (Step 3): Let $\mathcal{I} = \{I_1, \ldots, I_t\}$ be intersection results from Step 2. Sort any I_i, I_j as $I_i < I_j$, if $I_i \subseteq I_j$. For those I_i and I_j s.t. they do not contain in each other but both satisfy that $I_i, I_j > I'_i$ and $I_i, I_j < I_{j'}$, they are put between I'_i and I'_j, but without any order within themselves. For example, the sorted order might be $\{R_1 \cap R_2 \cap R_{new}, R_1 \cap R_3 \cap R_{new}\} < R_2 \cap R_4 \cap R_{new} < R_5 \cap R_{new}$, in which the order between $R_1 \cap R_2 \cap R_{new}$ and $R_1 \cap R_3 \cap R_{new}$ is not decidable. Let the sorted result be S_2. It is straightforward that unlike S_1 (in Step 1) that has a total order, S_2 only has a partial order.
Final merge (Step 4): We merge S_1 and S_2 as $S = \{S_2, R_{new}, S_1\}$. Obviously all elements in S_2 is contained in R_{new} as well as all elements in S_1. Then starting from the first element in S, we apply Case 1 (if the intersection equals R_{new}), Case 2 (if the intersected rules are contained in R_{new}), and Case 4 (if the rules only intersect with R_{new} on each intersected result in S_2). For the rules with undecided orders, their key assignments are independent from each other. After finished S_2, we apply Case 3 on each rule in S_1, following their orders in S_1.

From the four construction steps, we have:
Lemma 2. *Given the original AC rules $\mathcal{R} = \{R_1, \ldots, R_n\}$ and the new rule R_{new}, the construction procedure preserves the monotone property of the keys of $\mathcal{R} \cup \{R_{new}\}$, i.e., for any two rules $R_i, R_j \in \mathcal{R} \cup \{R_{new}\}$, let K_i and K_j be the keys of R_i and R_j, then if $R_i \subseteq_{ac} R_j$, then $K_i \subseteq K_j$.*

Finally, we state the following theorem.
Theorem 1. *The above construction procedure is deterministic.*

4 Experiments

We conducted various experiments to gauge the effectiveness of our approach under various network configurations. We setup a P2P network by using FreePastry, a DHT-based P2P network simulation testbed. Our algorithms were implemented in *Java* and the experiments were run on an Intel Core 2 Duo, 2.4GHz, Windows machine equipped with $4GB$ main memory. Every result is the average of about 10 runs. Due to the space limit, more details about the experiments, including the AC rules and the queries used, can be found at the following URL [18].

4.1 Setup

We setup several networks, with size ranging from 100 to 1500 peers. For each network, we vary the percentage p = 1%, 5%, 10%, 25%, 50%, 100% of nodes that share the key pieces. We employ the (m, n) threshold scheme by Shamir [11] in our experiments. For each setup of available key pieces, we vary the m values, i.e., the number of key pieces needed for reconstruction. To test the performance of our approach, we use three measurements of the time: (1) *SST*, the secret sharing time, which measures the time needed for generating and distributing the key pieces, (2) *SRT*, the secret reconstruction time, which measures the time needed for reconstructing the decryption key, and (3) *NL*, the network latency, indicating the communication cost due to the underlying network.

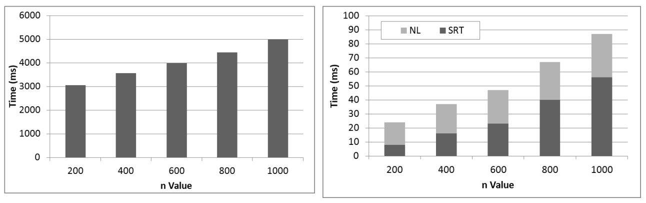
To measure the query evaluation performance over encrypted data, we use the TPC-H[3] benchmark dataset. We use MySQL 5.1 as the query engine. In our experiment, each node in the network stores locally a portion of the dataset. We design two schemes to vary the size of the local dataset, the *uniform distributed* scheme that evenly distributes the dataset to all nodes in the network, and the *randomly distributed* scheme that assigns local repositories of different sizes to the nodes. Furthermore, to measure the impact of the AC rule configuration, we setup five sets of AC rules (each set including 50 AC rules) that cover 20%, 40%, 60%, 80%, and 100% of nodes in the network. Typically, a query is asked locally on a node, and needs to be answered on all other nodes in the network. The query performance time has been considered as a composition of four times: the local query evaluation time, SRT, NL and the decryption time.

4.2 Overhead of Key Management

First, we vary the value of n, the number of key pieces, and use $m = [(n+1)/2]$, which is the number of needed key pieces for key reconstruction for the worst case. Figure 2 (a) & (b) show the measurement of SST, SRT and NL. We observe that SST, SRT and NL time grow with larger n values. Moreover, the SST time is always orders of magnitude larger than SRT time. This shows that while the initial setup takes time, the later key reconstruction procedure incurs little overhead, thus showing that the enforcement of access control by using distributed encryption is indeed of practical use.

we also measure the impact of m values on SST, SRT and NL. We vary m values from the worst case $m = [(n+1)/2]$ up to the total number of key pieces $m = n$. We observe that both SST and SRT time grow linearly with increasing m values, which is straightforward as more key shares are needed for

[3] http://www.tpc.org/tpch/



(a) SST (b) SRT

Fig. 2. Various n values; $m = [(n+1)/2]$; Network size: 1000 nodes.

key reconstruction. However, SST time increases with a more remarkable curve, as key distribution is a blind procedure that randomly chooses m peers, while key reconstruction is guided with index on the key distributee peers. We also observe that the NL time is not affected much by the variation of m values, as the keys have been distributed to a fixed number of peers; the communication time with these peers for key reconstruction is fixed. Due to the space limit, the results are omitted.

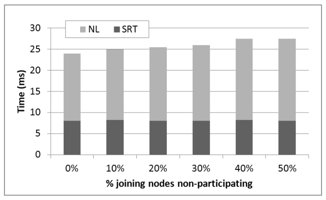
4.3 Network churn

We simulate the join and leave of peers in our framework. Figure 3 shows the result of inserting new nodes. We consider two possible configurations, the newly inserted peers/old leaving peers are/are not among the participants of secret sharing. In the first case, when the new joining peers do not share the key pieces (Figure 3 (a)), SRT is not affected much, while the network latency slightly increases. Notice that SST time is not reported in this case since the distributed key pieces are not recomputed. In the second case, when the new joining peers participate in secret sharing (Figure 3 (b) & (c)), SST, SRT and network latency increase with larger number of such peers. In particular, the increase of SST is more significant than that of SRT and network latency, since the secret has to be recomputed. However, SST has still acceptable values for both schemes, thus confirming that the secret computation overhead is negligible.

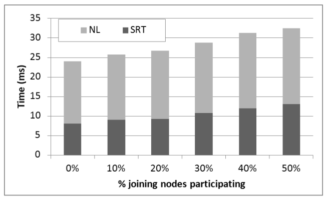
We observe the similar results for leaving peers. First, when the leaving peers do not share the key pieces SRT does not change much, while the network latency slightly decreases, and SST stays the same for this configuration. Second, when the leaving peers participate in secret sharing, SST, SRT and network latency decrease with larger number of such peers. Finally, comparing the impact of leaving/joining peers to both SRT and NL, we observe that the former is less than the latter. The above trend is due to that fact that, in the case of leaving peers the number of participants sharing the secret is reduced, while, in the case of joining peers, such number is comparably increased. Due to space limit, the details of the results are omitted.

4.4 Query Evaluation

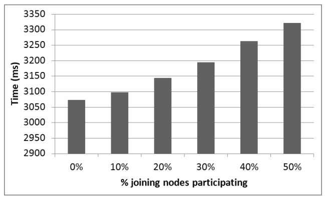
We start from the uniform distributed scheme, and measure the query performance in various cases. First, we vary the total number of tuples in the network from $100K$ to $500K$, while keeping the coverage of AC rules constant. Figure 4 (a) presents the results for TPC-H datasets. It can be observed that the SRT is



(a) SRT: # of new peers that do not share the key

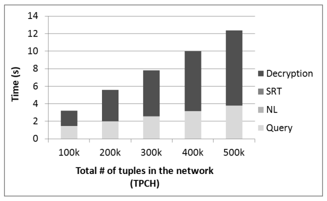


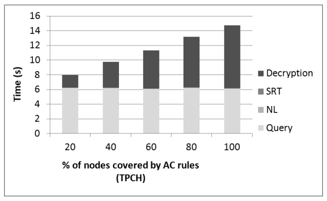
(b) SRT: # of new peers that share the key



(c) SST: # of new peers that share the key

Fig. 3. Join of peers; $n = 200$; $m = [(n + 1]/2$; Network size: 1000 nodes.



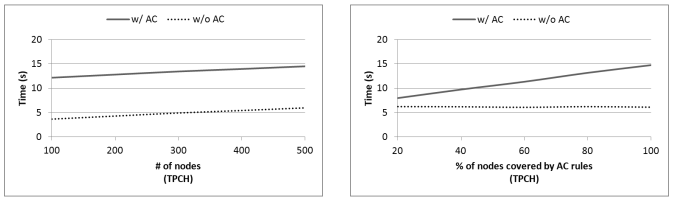


(a) Various total # of tuples; 100 nodes. (b) Various coverage of AC rules; 500 nodes, 500k tuples.

Fig. 4. Query performance on uniform distributed data

relatively stable thus confirming that key reconstruction is independent of the underlying databases sizes; by opposite, local query evaluation time, NL, and decryption time increase for larger databases. Furthermore, local query evaluation time and decryption time are dominant for all database sizes. Then, we vary the coverage of AC rules in the network, while keeping the number of tuples as $500K$. Figure 4 (b) shows that the query evaluation time does not change much, as the queries are always evaluated on the whole network. However, the decryption time increases when more nodes are covered by AC rules; the increase is linear to the increase of number of nodes that are covered by AC rules. This happens because the number of tuples that are encrypted is linear to the number of AC rules (recall that each node has the same number of tuples). Nevertheless, the increase is not overwhelming; even for the case that 100% nodes are covered, the total time for query evaluation (including decryption, SRT and NL) is only around 14 seconds. In other words, the overhead incurred by the AC rule configuration and enforcement is reasonably low. We also vary the network size from 100 nodes to 500 nodes and measure the query evaluation time. We observe that query evaluation time and decryption time are relatively stable with regard to the network size, since the queries are evaluated on the same size of data in the network. For the sake of conciseness, we omit the results.

Then, we rerun the same experiments under a randomly distributed scheme. The experiment results are similar to those of uniform distributed data. The only difference we have observed is that when we vary the number of nodes that are covered by AC rules, the increase of query evaluation time is not relatively linear to the increase of number of nodes that are covered by AC rules, since the data on these nodes are of different sizes. Due to the space limit, the results are omitted. In Figures 5, we have compared the query performance for both cases with and without access control (w/ AC and w/o AC, respectively; the latter being the case in which no data is encrypted). We observe that the results are not



(a) Various # of nodes; AC rules Coverage: 100%.

(b) Various coverage of AC rules; 500 nodes.

Fig. 5. Query performance comparison(w/ AC: with access control, w/o AC: without access control); total # of tuples: 500k

affected much by the increase of the network size (Figure 5 (a)). These results show that the cost for applying our algorithms to protect the access to selected items in a distributed scenario is affordable, thus confirming their utility and efficiency. However, query performance grows with increase of AC rules coverage. In particular, Figure 5 (b) shows that with small AC rules coverage (e.g., 20%), access control enforcement only incurs the overhead as around 25% of the query evaluation time needed for w/o AC case; with increasing AC rules coverage to 100%, the overhead of enforcing AC rules becomes around 150% of the query evaluation time (TPC-H dataset, Figure 5 (b)) for the w/o AC rules case. This is the price that we need to pay for the enforcement of access control policies.

5 Related Work

Building distributed persistent storage has been the goal of previous file-sharing projects, such as OceanStore [5], Plutus [6] and Cryptree [19]. As opposed to our approach, all the above systems work in a client-server architecture, and do not scale for a large group of users. Even if with varied features, omitted for the sake of space, they rely on a common centralized authorization authority.

Access control over replicated databases is studied in [20], where a security mechanism based on secret sharing is enforced in the presence of quorum servers (pair-wise replicated servers with overlapping information). The underlying assumption is that the users might be untrusted, while the servers are all trustworthy.

Enforcing access control by using cryptography have been studied in various contexts (e.g., XML data publishing [8] and distributed file systems [10]). However, none of them would be applicable to PDMS, as they either do not need a distribution-aware policy language, or rather employ a one-to-one key assignment from the data owner to the user, as it is common in data publishing [8]. Furthermore, distributed key management has been studied in the network context (e.g., mobile ad-hoc networks [21] and ad-hoc wireless networks [16]). Such networks are characterized by peers with low bandwidth, intermittent network connectivity and scarcity of computational resources, which is not the case for PDMS [1], where each peer is a database. To the best of our knowledge, the problem of distributed and resilient access control in such databases has not been tackled before [3, 2, 1]. The literature on cryptographic access control to address the problem of cost effective key management has been studied in the context of the Web [22, 23]. In our work, we focus on distributed access control in P2P networks, and discuss the extensions needed in such scenarios where previous key management approaches [22–24] are not directly applicable.

The most recent P2P algorithms realizing the efficient DHT (Distributed Hash Tables) abstraction [25, 26] are vulnerable to misbehaving nodes, and the only measure adopted in [26] is to randomize the routing procedure in order to avoid the 'bad' nodes. However, this would not be tolerated in a PDMS, where robust database access control is of utmost importance. To the best of our knowledge, the only work that deals with the distribution of privilege enforcement in P2P networks is PACS [7]. However, they rely on role-based access control, where the access policy is determined by the system, not by the data owner. In this work, we focus on discretionary access control, and the capability of deciding the access granularity by using SQL-based policy rules. Bertino et al. [27] aimed at defining an extension of XACML access rules on resources located in large-scale distributed systems. They focus on the problems of integrating conflicting policies in such a setting. DTD secure views for XML data have been defined in [28]. The approach, based on DTD annotations and view-based, would not be applicable to our context.

6 Conclusions and Future Work

In this paper, we studied the problem of distributed and resilient access control in P2P databases. In particular, we adapted secret sharing to PDMS, devised a block-based encryption scheme that supports overlapping AC rules with shared access to the same data, and proposed a solution for the efficient enforcement of updates on AC rules. As a further goal, we plan to investigate the impact of heterogeneous schemas in PDMS, and the secure query reformulation strategies in such a distributed resilient paradigm.

References

1. Steven D. Gribble and Alon Y. Halevy and Zachary G. Ives and Maya Rodrigand and Dan Suciu: What Can Database Do for Peer-to-Peer? In: Proc. of WebDB. (2001)
2. Hose, K., Roth, A., Zeitz, A., Sattler, K.U., Naumann, F.: A research agenda for query processing in large-scale peer data management systems. Inf. Syst. **33**(7–8) (2008) 597–610
3. Bonifati, A., Chrysanthis, P.K., Ouksel, A.M., Sattler, K.U.: Distributed databases and peer-to-peer databases: past and present. SIGMOD Rec. **37**(1) (2008) 5–11
4. Sandhu, R., Zhang, X.: Peer to peer access control architecture using trusted computing technology. In: Proc. of ACMT. (2005)
5. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: Oceanstore: an architecture for global-scale persistent storage. SIGPLAN Not. **35**(11) (2000) 190–201
6. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable secure file sharing on untrusted storage. In: Proc. of FAST. (2003)

7. Sturm, C., Hunt, E., Scholl, M.H.: Distributed privilege enforcement in pacs. In: DBSec. (2009) 142–158
8. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: Proc. of VLDB. (2003)
9. Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Sarnarati, P.: Securing xml documents. In: Proc. of EDBT 2000
10. Harrington, A., Jensen, C.: Cryptographic access control in a distributed file system. In: Proc. of ACMT. (2003)
11. Shamir, A.: How to share a secret. In: Comm. of the ACM, Vol. 22, Nr. 11. (1979) 612–613
12. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. In: Proceedings of ICDT. (1998) 56–70
13. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. In: Proc. of STC. (1977)
14. Saraiya, Y.P.: Subtree-elimination algorithms in deductive databases. In: Thesis, Stanford University. (1991)
15. 5200.28-STD, D.S.: Trusted Computer System Evaluation Criteria. USA Dept. of Defense. (1985)
16. Luo, H., Lu, S.: Ubiquitous and robust authentication services for ad hoc wireless networks. Technical report, University of California, Los Angeles (2000)
17. Joshi, D., Namuduri, K., Pendse, R.: Secure, redundant, and fully distributed key management scheme for mobile ad hoc networks: an analysis. EURASIP J. Wirel. Commun. Netw. (4) (2005) 579–589
18. : P2Pac Web Site. `http://staff.icar.cnr.it/angela/p2pac/exp/exp.html`
19. Grolimund, D., Meisser, L., Schmid, S., Wattenhofer, R.: Cryptree: A folder tree structure for cryptographic file systems. RDS (2006) 189–198
20. Naor, M., Wool, A.: Access control and signatures via quorum secret sharing. IEEE TPDS **9**(9) (1998) 909–922
21. Merwe, J.V.D., Dawoud, D., McDonald, S.: A survey on peer-to-peer key management for mobile ad hoc networks. In: ACM Comp. Surveys. (2007)
22. Kayem, A.V.D.M., Akl, S.G., Martin, P.: On replacing cryptographic keys in hierarchical key management systems. Journal of Computer Security (16(3)) (2008) 289–309
23. Sun, Y.L., Liu, K.J.R.: Analysis and protection of dynamic membership information for group key distribution schemes. IEEE Transactions on Information Forensics and Security (2(2)) (2007) 213–226
24. Blundo, C., Cimato, S., di Vimercati, S.D.C., Santis, A.D., Foresti, S., Paraboschi, S., Samarati, P.: Efficient key management for enforcing access control in outsourced scenarios. In: Proceedings of SEC. (2009)
25. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of SIGCOMM. (2001)
26. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. of SIGCOMM. (2001)
27. Mazzoleni, P., Crispo, B., Sivasubramanian, S., Bertino, E.: XACML Policy Integration Algorithms. ACM TISS **11**(1) (2008) 1–29
28. Wenfei Fan, Chee-Yong Chan, M.G.: Secure xml querying with security views. In: Proc. of SIGMOD 2004