Log Auditing for Trust Assessment in Peer-to-Peer Collaboration

Hien Thi Thu Truong, Claudia-Lavinia Ignat

Log Auditing for Trust Assessment in Peer-to-Peer Collaboration

Hien Thi Thu Truong, Claudia-Lavinia Ignat
INRIA Nancy-Grand Est, France
{*Hien.Truong,Claudia.Ignat*}*@loria.fr*

***Abstract*—In order to overcome the disadvantages of a central authority, a tendency is to move towards a peer-to-peer collaboration where control over data is given to users who can decide with whom they want to share their private data. In this peer-to-peer collaboration it is very difficult to ensure that after data is shared with other peers, these peers will not misbehave and violate data privacy. In this paper, we propose a mechanism that addresses the issue of data privacy violation by auditing the collaboration logs. In our approach, trust values between users are adjusted according to their previous activities on the shared data. Users share their private data by specifying some obligations the receivers are expected to follow. We log modifications done by users as well as the obligations and use a log-auditing mechanism to detect users who misbehaved. We adjust their associated trust values by using any existing decentralized trust model.**

***Keywords*-P2P collaboration; log auditing; trust assessment; collaborative editing;**

I. Introduction

Social software including wikis, blogs, micro-blogs and social networks has emerged as a new interpersonal communication form. Existing micro-blogging services such as Twitter and social networks such as Facebook have millions of users using them everyday. While these social services offer many attractive functionnalities, they feature some limitations. Most of the platforms hosting these social services rely on a central authority and place personal information in the hands of a single large corporation which is a perceived privacy threat. Users must provide and store their data to vendors of these services and have to trust that they will preserve privacy of their data, but they have little control over the usage of their data after sharing it with other users. Moreover, user communities cannot deploy these kind of service applications since they generally rely on costly infrastructures rather than allowing sharing infrastructure and administration costs.

In the last decades peer-to-peer (P2P) technology received significant research attention and a widespread use in open-source software and industry communities. The main strengths of P2P systems [10] are their independence of a centralized control and of a dedicated infrastructure. Participating nodes are owned and operated by independent individuals and therefore administration costs of the system are shared. Distinctive characteristics of P2P systems are high scalability, resilience to faults and attacks and a low deployment barrier for new services. Due to all these properties, some recent approaches such as [1] proposed moving away from centralized authority-based collaboration towards a P2P trust network where control over data is given to users who can decide with whom to share their data. The risk of privacy breaches is decreased in this P2P collaboration as only part of the protected data is exposed at any time. However, it is very difficult to ensure that after data is shared with other peers, these peers will not misbehave and violate data privacy. Usage control mechanisms [8], [3] model policy or obligation that users receive together with data which refer to what happens to data after it has been released to authorized people, for example, how they *may*, *should* and *should not* use it. However, existing usage control approaches rely on some central authorities that can check violation of obligations. These solutions are not suitable for P2P collaboration where no peer has control over another peer to audit his behavior and a peer can observe only behavior of collaborating peers. To prevent data misuse in P2P environments, trust management mechanisms are used where peers are assigned trust values according to past experiences with other peers. According to assigned trust values, a peer may choose to collaborate only with trusted peers.

In this paper, we propose a novel trust management mechanism adapted for P2P collaborative editing. Each peer maintains a local workspace that contains local data as well as modifications done on the shared data and obligations related to usage policy of the shared data. The logged modifications and obligations are shared with other peers. Each peer performs a log auditing mechanism for detecting misbehaved peers and adjusts their trust values according to audit result.

The paper is structured as follows. In Section II we start by presenting a motivating example. In Section III we go on by giving an overview of the collaboration model that we propose. We describe the formal log structure and logging mechanism in Section IV. We provide definition of ordering obligations and our merging logs mechanism in Section V. Section VI presents our indications of misbehavior, algorithm for detecting cheater and local trust adjustment. We discuss related work in Section VII and Section VIII concludes this paper.

II. Motivating example

Let us consider a data sharing motivating example in a collaborative environment. Suppose Alice creates a document and she wants to share it with different friends, say Bob and Carol. She shares it with Bob with the permission *"may add comment"* on the document but with the prohibition *"should not modify"* it.

In a decentralized environment, it is very difficult to guarantee Bob will not misbehave on the document after Alice shared it with him. In our approach, we propose a mechanism of logging past actions of users concerning shared data. When a user shares the data with other users the log of actions done on the shared data is disclosed as well. Thus if Bob shares the document with Carol, she can therefore check the actions of Alice and Bob on the shared document. If Carol shares further the document with Alice, Alice can check the actions of Bob and Carol and verify if Bob misbehaved on the document.

Our system does not aim to prevent fraud. Rather, the log mechanism provides audit capabilities in order to detect attempts at fraud after the data has been shared and used. The local actions on data and the communications between peers are assumed to leave some evidence and hence are observable. Users attach a usage policy to the data in order to specify what actions are allowed and under which conditions. Modifications done by users on the shared data as well as the obligations that must be followed are logged. An a-posteriori checking for compliance to obligations can be performed by each user when he receives a shared data by auditing the log. Users locally evaluate trust on other users according to results of log auditing mechanism. Computed trust values help users decide to continue or respectively initiate collaboration with other users.

III. P2P collaborative editing with obligations

Several works in the domain of distributed collaborative editing exist as Git[1], Darcs[2], Logoot[14]. These systems allow a group of users edit the same text/graphics/multimedia documents. In order to achieve high responsiveness, high concurrency and good fault-tolerance, documents are replicated at each peer site. A peer can perform locally modifications at any time. Then these local modifications are propagated to other peers. A P2P collaborative editing system should not only respect the CCI (Causality, Convergence, Intention preservation) criteria, but also support P2P constrains (scalability, churn, unknown number of peers, failures). We enhanced the current model of P2P collaborative editing that exchanges and merges modifications performed by users with obligations associated with shared data. We briefly present below an overview of the proposed collaboration model:

- As in the general model of P2P collaborative editing, peers log editing modifications performed on shared documents. When peers synchronize their work with other peers the modifications logs are exchanged rather than full document states. Document states can be computed by executing operations in the log. In addition to document modifications we log also obligations associated to the shared document when modifications are sent to other peers. For simplicity throughout this paper we will sometimes specify that users share documents, but in fact they share the logs of modifications done on the document together with the received obligations.
- When a peer receives a log of modifications and obligations from another peer, it merges the remote log with the local log. Merging algorithms have to deal not only with conflicts between modifications but also with conflicts between obligations. We use a CRDT (Commutative Replicated Data Type) approach [14] for merging document modifications where operations are designed to commute and therefore any execution order of operations would lead to the same document state. However, merging approach maintains a causal order between modifications and obligations in order that the validity range of obligations can be determined.
- Peers are expected to behave correctly, but they might be suspected of incorrect behavior. Log auditing mechanism detects misbehavior of cheaters. A peer has initial trust values associated with other peers. They are calculated and adjusted after each log analysis.

IV. Logging mechanism

In this section, we present the log structure and give an example to illustrate how log is created and stored locally at sites of peers.

Definition 1 (Event)**.** An event e is defined as a triplet of (*type*, *operation*, *attributes*), where:

- *type* $\in \mathcal{T}$, $\mathcal{T}$ = { *action*, *may*, *may not*, *should*, *should not* }, *action* denoting a peer action, and *may*, *may not*, *should*, *should not* referring to a specific obligation ;
- *operation* $\in \mathcal{S}$, $\mathcal{S}$ being the set of modifications that can be produced by peers ;
- *attributes* being the set of event parameters in the form of {attribute name, attribute value} pairs.

As an example, suppose that $\mathcal{S}$ = {create, insert, delete, share}. An event e_1 = (*action*, *insert*, {by, P_1}) means that P_1 performs an action of insertion. An event e_2 = (*should share*, {by, P_1}, {to, P_2}) denotes that P_1 gives P_2 an obligation to share the document.

Definition 2 (Log)**.** A log L is an append-only ordered list of events in the form $[e_1, e_2, \ldots, e_n]$. The log maintains the property that events generated in causal order will appear in this order in the log.

[1]http://git-scm.com/

[2]http://darcs.net/

The following rules are used for logging:

- Each new event generated locally is added to the end of local log in the order of occurrence.
- The events from remote log that should be merged are added to the end of local log in the order they appeared in the remote log.
- When a peer shares the document with other peers the *share* events together with the specified obligations are logged by the receiving peer. However, we make the assumption that a peer is unwilling to disclose with all collaborating peers all the sharing events and obligations that he specifies to a certain peer. That is why sharing events and obligations are not kept in the local log of the sending peer. For instance, if a peer A shares document with peer B and afterward with peer C, peers B and respectively C are not aware of sharing actions and obligations peer A gave to C and respectively B. If we model peers as nodes in a directed graph and sharing events with their associated obligations as directed edges from source to destination, a peer is aware of only those share events that are present on the sharing path from the peer who created the document to the current peer.

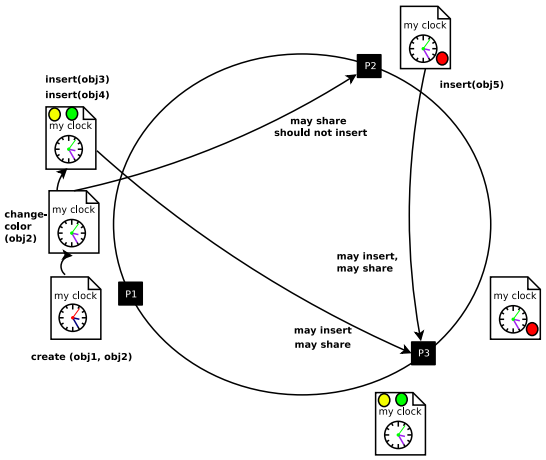

Figure 1. An example of collaboration

Figure 1 shows an example of collaborative editing in a distributed P2P network between three peers P_1, P_2, P_3. The set of operations $\mathcal{S}$ that peers can perform is {*create*, *insert*, *delete*, *change-color*, *share*}. The set $\mathcal{T}$ of event types is {*action*, *should*, *should not*, *may*, *may not*}. Let P_1 be the creator of document d. P_1 shares d with P_2, and then P_2 after performing some additional changes shares d with P_3. In parallel, P_1 after performing some additional changes on d shares it with P_3. The log is created locally at peers as follows:

1) At the local site, P_1 creates a document d with two objects *obj1* and *obj2*. It changes color of object *obj2* and shares the document with P_2 with obligations *"may share"* and *"should not insert"*. Then, P_1 inserts objects *obj3* and *obj4* and sends the document to P_3 with obligations *"may share"* and *"may insert"*. Notice that the obligations given by P_1 to P_2 and P_3 are different. The log of P_1 is shown in Table I.

Table I
EVENT LOG OF P1

No.	Event		
	type	*operation*	*attributes*
1	action	create	{by,P_1}
2	action	insert	{by,P_1} {object,obj1}
3	action	insert	{by,P_1} {object,obj2}
4	action	change-color	{by,P_1} {object,obj2} {color,blue}
5	action	insert	{by,P_1} {object,obj3}
6	action	insert	{by,P_1} {object,obj4}
...			

2) P_2 receives d from P_1 with obligations *"may share"* and *"should not insert"*. The log received from P_1 (entries $1-4$ in Table II) is updated by appending the *share* event from P_1 (entry 5 in Table II) and the received obligations (entries 6 and 7 in Table II). Further P_2 inserts an object *obj5* to d. Thus, the log is updated by adding the insert event (entry 8 in Table II). Note that P_2 received document d with obligations *"may share"* and *"should not insert"* and he violated these obligations as he performed an insertion. The log of P_2 is shown in Table II.

Table II
EVENT LOG OF P2

No.	Event		
	type	*operation*	*attributes*
1	action	create	{by,P_1}
2	action	insert	{by,P_1} {object,obj1}
3	action	insert	{by,P_1} {object,obj2}
4	action	change-color	{by,P_1} {object,obj2} {color,blue}
5	action	share	{by,P_1} {to,P_2}
6	may	share	{by,P_1} {to,P_2}
7	should not	insert	{by,P_1} {to,P_2}
8	action	insert	{object,obj5}
...			

3) P_3 receives d from P_1 and afterward from P_2. P_3 can detect misbehavior of P_2. P_3 still can accept the document from P_2, but in this case P_3 has to merge the two document versions received from P_1 and P_2. The log of P_3 after merging is shown in Table III.

Table III
EVENT LOG OF P3

No.	Event		
	type	*operation*	*attributes*
1	action	create	{by, P_1}
2	action	insert	{by,P_1} {object,obj1}
3	action	insert	{by,P_1} {object,obj2}
4	action	change-color	{by,P_1} {object,obj2} {color,blue}
5	action	insert	{by,P_1} {object,obj3}
6	action	insert	{by,P_1} {object,obj4}
7	action	share	{by,P_1} {to,P_3}
8	may	insert	{by,P_1} {to,P_3}
9	may	share	{by,P_1} {to,P_3}
10	action	share	{by,P_1} {to,P_2}
11	may	share	{by,P_1} {to,P_2}
12	should not	insert	{by,P_1} {to,P_2}
13	action	insert	{by,P_2} {object,obj5}
14	action	share	{by,P_2} {to,P_3}
15	may	insert	{by,P_2} {to,P_3}
16	may	share	{by,P_2} {to,P_3}
...			

In order to detect cheaters, each peer analyzes the received logs. A peer whose actions do not conform to obligations is considered as a cheater. In the above example, P_3 detects the violation of insertion made by P_2.

When a peer considers merging remote logs to the local log, it needs to check firstly the trust value associated to the remote peer. If the remote peer can be trusted, conflicts between local and remote obligations have to be detected and a decision has to be taken if merging should performed or not. In case of a positive merging decision, merging of modifications and obligations is performed. In the previous example for a better understanding of the logging mechanism we considered no conflicts between obligations and modifications when merging was performed by P_3.

In the next section we proceed with a presentation of the log synchronization mechanism which deals with resolving conflicts between obligations and document modifications.

V. LOG SYNCHRONIZATION

In this section we present the mechanism for synchronization of logs among peers. Before presenting the algorithms for merging document content and obligations, we describe the formalization of obligations and the solution that we adopt for ordering obligations.

Obligation represents an action that has to be fulfilled in the future. We use the modalities *should* and *may* to express obligations, where *should* implies commitment, while *may* expresses the permission to the referred action. An obligation is denoted by a modality followed by an action. To express negation of an obligation, a negative adverb *not* is inserted before the action (e.g. *"may not write"* or *"should not share"*).

Two main types of conflicts exist between obligations are:

- conflicts between firm commitments and permissions referring to the same operation: between *may*[operation] and *should not*[operation] and between *may not*[operation] and *should*[operation]
- conflicts between firm commitments and negation of firm commitments referring to the same operation: between *should* [operation] and *should not* [operation]

The challenge is to prevent contradictions eventually by reaching a compromise consistent for all obligations a peer currently holds. For the moment we consider a limited set of operations that can be performed for collaboratively editing documents and we do not consider hierarchies between obligations defined on these operations. For instance, *"may write"* and *"may share"* have no hierarchy order. Thus, some priorities can be established for resolving conflicts in terms of specific objectives and depending on the collaborative system. For instance, we can consider that *may* has higher priority than *should* in order to give more freedom to users for performing actions because *should* implies a commitment while *may* does not. Besides ordering single obligations by using priorities, we propose a mechanism for ordering sets of obligations called composite obligations.

Let $\mathcal{S}$ be a set of n operations that could be ordered [op_1, op_2, ..., op_n] with $op_1 > ... > op_n$, $\mathcal{R}$ be a set of n-digit binary numbers from 0 to 2^n and a composite obligation $\mathcal{O}$ linked to operations of $\mathcal{S}$. Obligation $\mathcal{O}$ could be mapped to n-digit number m of $\mathcal{R}$ by setting each digit of m as:

- if $\{may,\ may\ not\}\ op_i \in \mathcal{O}$, the i^{th} digit of m is 1;
- if $\{should,\ should\ not\}\ a_i \in \mathcal{O}$, the i^{th} digit is 0;
- if $\{may,\ may\ not,\ should,\ should\ not\}\ op_i$ is not specified in $\mathcal{O}$, the i^{th} digit of m is $*$. Note that mapping rule also has to deal with the case an obligation is not defined, thus without losing the generality we define $1 > * > 0$ for digit-based comparison.

The comparison of two composite obligations $\mathcal{O}_1$ and $\mathcal{O}_2$ with corresponding number m_1 and m_2 turns to the comparison of m_1 and m_2 that $(\mathcal{O}_1 > \mathcal{O}_2) \Leftrightarrow (m_1 > m_2)$.

For instance, given a set $\mathcal{S}$ of two operations (*n=2*) op_1, op_2 with the order $[op_1, op_2]$ and two obligations $\mathcal{O}_1$ = $[should\ op_1, may\ op_2]$ and $\mathcal{O}_2$ = $[should\ op_2]$ generated over $\mathcal{S}$. The n-digit numbers m_1 = "01" and m_2 = "$*$0"

Algorithm 1: Check for Merge

Input: L_v^{remote} - remote log received from v,
L_u^{local} - local log of u,
$\mathcal{O}_u^{local}$ - the set of local obligations of u,
$\mathcal{O}_u^{remote}$ - the set of obligations of u received from v,
$Trust_u(v)$, θ - minimum trust value required
Output: *tobeMerged* $\in$ {*NoMerge-NoBranch*, *NoMerge-Branch*, *Merge-NoBranch*}

```
1  begin
2      if Trust_u(v) < θ then
3          return NoMerge-NoBranch;
4      end if
5      else
6          O_1 ← O_u^local; O_2 ← O_u^remote
7          foreach o ∈ O_2 do
8              if ∃o' ∈ O_1|(o⊥o') ∧ (o not overrides o') then
9                  return NoMerge-Branch;
10             end if
11         end
12         return Merge-NoBranch;
13     end if
14 end
```

Algorithm 2: Merge

Input: L_v^{remote} and L_u^{local}, *tobeMerged*
Output: new L_u^{local}

```
1 begin
2     if tobeMerged = Merge-NoBranch then
3         for event e_k ∈ L_v^remote do
4             if e_k ∉ L_u^local then
5                 append e_k to the end of L_u^local;
6             end if
7         end for
8     end if
9 end
```

are mapped from $\mathcal{O}_1$, $\mathcal{O}_2$ to $\mathcal{R}$ over $\mathcal{S}$. Since $m_2 > m_1$, then $\mathcal{O}_2 > \mathcal{O}_1$ ([*should* op_2] > [*should* op_1, *may* op_2]).

A peer can receive a remote log with conflict obligations. Since it cannot merge conflicts, either it rejects or creates a new branch for the remote version or it leaves the local version to accept new one. If the peer is willing to share the document with other peers, it cannot give higher obligations than what it currently holds. In Algorithm 1, a peer will not merge or create new branch if the sender is distrusted. At line 8 of algorithm, we have the condition to create new branch if the remote log includes at least one obligation which conflicts with current obligations. We consider overriding of obligation if a peer revokes an old obligation and replaces it by new one. For instance, the old obligation *"should not share"* which peer v received from peer u can be overridden by new one *"may share"*. Line 12 returns the result that two logs can be merged since there is no conflict found.

When a peer u receives changes from a remote peer v we use an anti-entropy mechanism [9] to detect the new events from the remote log L_v^{remote} that have to be integrated into the local log L_u^{local}. The new events are appended to the end of current local log L_u^{local}. The complexity of this checking mechanism is $O(m \times n)$ where m and n are the sizes of L_u^{local} and respectively L_v^{remote}. The anti-entropy mechanism preserves the causal relations between operations in the log.

The merging mechanism that we used is based on CRDT approach in which concurrent operations can be replayed in any order as operations are designed to commute. Therefore the merging is simply performed as shown in Algorithm 2. CRDT ensures convergence of document replicas at different sites if the same set of operations is executed at those sites independently of their execution order.

The decision of acceptance or rejection of coming document is essentially based on the comparison of composite obligation that the peer currently holds with the composite obligation that the peer might receive. If remote changes are accepted, the merging mechanism puts all new events to the end of local log. Note that new received obligations updated to log should not conflict with current obligations. Only the owner of obligations has the right to override obligation by making it more or less restrictive. In addition, a peer decides to merge or not a remote log with the local one depending on the trust value of sender. If the sender is distrusted, the peer might reject remote log.

VI. Trust assessment

In this section we describe our proposed solution for assessing trust based on log auditing. We first present our log auditing algorithm for detecting cheaters and then discuss how trust values are adjusted based on the auditing results.

We consider a collaborating system where each peer is supposed to respect given obligations. If it does, then we call the peer *trusted*; otherwise we call the peer *distrusted* or *suspected*. There are two ways in which a peer cannot be *trusted*: it can either do actions violating obligations or ignore an obligation. Ideally, if a peer misbehaves in either of these ways, other peers should detect its misbehavior. A peer is considered as distrusted if it violates an obligation, and a peer is considered as suspected if it does not prove that it conforms to a commitment obligation. For instance, a peer that receives the obligation *"should send back"* the document, but it never fulfills this obligation is considered suspected. Note that peer u withdraws the suspected indication on peer v when it finds that peer v fulfilled the obligation he was suspected for.

Cheaters may try to hide their misbehavior by modifying the log. Briefly, a peer u is considered $cheater$ if it modifies the log that consequently affects auditing result. For instance, u removes some obligations that it does not want to be enforced to follow. The log auditing mechanism should guarantee also that peers cannot modify the log. To prevent log modification, we use authenticators for patches

Algorithm 3: Log Auditing

Input: L - the current local log of assessing peer u,
$\mathcal{O}_v$ - current obligation of each peer v appearing in L,
C_v - set of suspected obligations of each peer v.
Output: audit[v] for each peer v appearing in L.

```
1  begin
2      k ← 1;
3      while k ≤ lengthOf(L) do
4          if e_k.type = action then
5              op ← e_k.operation, v ← e_k.by;
6              if op violates O_v then
7                  audit[v] ← distrusted;
8              end if
9              if op conforms a should obligation ob then
10                 Remove ob from C_v;
11                 if (audit[v] = suspected)∧(C_v = ∅) then
12                     audit[v] ← trusted;
13                 end if
14             end if
15         end if
16         else
17             ob ← e_k; v ← e_k.to;
18             if ob ∉ O_v then
19                 Add ob to O_v;
20                 if ob overrides existing obligations ob' ∈ O_v
                   then
21                     Remove ob' from O_v;
22                 end if
23                 if e_k.type = should then
24                     Add ob to C_v;
25                     if audit[v] = trusted then
26                         audit[v] ← suspected;
27                     end if
28                 end if
29             end if
30         end if
31         k ← k + 1;
32     end while
33 end
```

of operations. Due to the space limitation, we do not present in this paper our solution about generation and verification of authenticators. Thus audit results are based on the assumption that log is maintained and shared correctly. The audit mechanism can generate only three types of results: *trusted*, *distrusted* and *suspected*.

The Algorithm 3 takes as input the local log of peer u that wishes detecting misbehavior of other peers. Peer u browses its own log and checks whether behavior of other peers is correct. It returns the list $audit$ that specifies for each peer in the log whether it is trusted, distrusted or suspected.

Peer u keeps a set of current obligations for each other peer v. During browse process, a new obligation can be added to current set of obligations of peer v. Note that in the algorithm, at the initial step, $audit[v] = \varnothing$, the set of obligations $\mathcal{O}_v = \varnothing$ and set of suspected obligation $C_v = \varnothing$. Some additional functions are required to check if an action violates a certain set of obligations or conforms to a commitment of obligation. In the auditing algorithm, the trust model is called when a violation is found. When auditing process finishes, a trust model is called to update audited results of all peers v. Log analyzing has polynomial order of n time complexity $O(n)$ with n is the size of log.

Checking a log is a basic mechanism to detect cheaters and help to predict the probability that they will continue cheat in future actions. In our auditing protocol, we use log to check whether the peer's behavior conforms to that of its obligation. The audit for each peer at once certain check results in one of three indications: *trusted*, *distrusted* and *suspected*. Next, the auditing result is mainly used to assess the trust value of the assessed peer.

The auditing protocol is performed at any time at local sites. We denote $Trust_u^{log}(v)$ as the trust value that a peer u assigns to peer v. All peers are set an initial default trust value. The peer u updates value $Trust_u^{log}(v)$ for peer v mainly based on the result of log analysis. In order to manage trust values for peer v, we can use an existing decentralized trust model. When an assessed peer v is detected as *distrusted* or *suspected*, its local trust value is recomputed by assessing peer u using a trust model. The total local trust values could be aggregated from log-based trust, reputation and recommendation trust. The trust computation to get total trust values varies from trust models. Details of trust model are not presented in this paper.

Our approach for trust assessment uses log auditing. The violation in case a cheater copies the content of a document in order to create a new one and then claims to be the document owner can not be detected by log auditing. However, the log could be used to discover the history of actions on document that helps to detect cheaters.

VII. Related work

In this section we first compare our work with access control solutions and then with some approaches that address data privacy in P2P systems. Then we continue by describing and comparing our proposed mechanism with other approaches that use some related solutions to our approach but in different contexts.

Our approach based on trust management is different from access control mechanisms. Instead of enforcing granted rights a-priori as in traditional access control mechanisms, we check obligations a-posteriori once data has been shared. Similarly, there exist some optimistic access control approaches [13] that check a-posteriori access policies. In these approaches, if user actions violate granted rights, a recovery mechanism is applied and all carried-out operations are removed. Usually, this recovery mechanism requires a centralized authority that ensures that the recovery is taken by the whole system. However, the recovery mechanism is difficult to be applied in P2P system where a peer does not have knowledge of the global network of collaboration. Generally, access control mechanisms aim at ensuring

that systems are used correctly by authorized users with authorized actions. Rather than ensuring a hard system security, we adopt a flexible trust management mechanism that helps users collaborate with other users they trust. Once cheaters are detected, people will reconsider carefully whether to collaborate with them in the future.

In order to return data ownership to users rather than to a third party central authority, some recent works [1], [15] explore the coupling between social networks and P2P systems. In this context privacy protection is understood as allowing users to encrypt their data and control access by appropriate key sharing and distribution. Our approach is complementary to this work and refers to what happens to data after it has been shared.

Another approach that addresses data privacy violation in P2P environments is Priserv [4], a DHT privacy service that combines the Hippocratic database principles with the trust notions. Hippocratic databases enforce purpose-based privacy while reputation techniques guarantee trust notions. However, this approach focuses on a database solution, being limited to relational tables. Moreover, as opposed to our solution, the Priserv approach does not propose neither a mechanism of discovering the malicious users that do not respect the obligations required for using the data nor an approach for updating the trust values associated to users.

OECD (Organization for Economic Cooperation and Development) defined basic privacy principles including: collection limitation, data quality, purpose specification, use limitation, security safe, openness, individual participation, accountability. We consider data privacy in collaborative working from the point of *use limitation* that users will specify how their data may and may not be used. We consider the decentralized system where documents are exchanged and shared among users. When a user receives a document, he is expected to work on the document by respecting obligations. The log mechanism is used to detect cheaters who do not respect their obligations. The formal framework in [3] allows specification of obligations. They present different mechanisms for checking adherence to commitments. However, all their proposed solutions are based on a central reference monitor that can ensure that data protection requirements are adhered to. As opposed to our approach, these solutions are not suitable for P2P environments where there is no central authority.

Keeping and managing event logs is frequently used for ensuring security and privacy. This approach has been studied in many works. In [2], a log auditing approach is used for detecting misbehavior in collaborative work environments, where a small group of users share a large number of documents and policies. In [6], [11], a logical policy-centric for behavior-based decision-making is presented. The framework consists of a formal model of past behaviors of principals which is based on event structures. However, these models require a central authority that has the ability to observe all actions of all users. This assumption is not valid in a P2P setting. Our proposed log-auditing mechanism works for a P2P collaboration and its complexity compared to the centralized solution comes from the fact that each peer has only a partial overview of the global collaboration and can audit only peers with whom it collaborates. Therefore, a peer can take decisions only from the information it possesses from the peers with whom it collaborates.

Trust management is an important aspect of the solution that we proposed. The concept of trust in different communities varies according to how it is computed and used. Our work relies on the concept of trust which is based on past encounters [7]. Various trust models for P2P systems exist such as NICE model [12], EigenTrust model [5] and our mechanism for discovering misbehaved users can be coupled with any existing trust model in order to manage user trust values.

VIII. Conclusion

Our vision is to replace central authority-based social software collaboration with a distributed collaboration that offers support for decentralization of services. In this context, our paper addressed the issue of data privacy violation due to data disclosure to malicious peers in a P2P collaboration. In our collaboration model users share their private data by specifying some obligations the receivers must follow. Modifications done by users on the shared data and the obligations that must be followed when data is shared are logged in a distributed manner. A mechanism of distributed log auditing is applied during collaboration and users that did not conform to the required obligations are detected and therefore their trust value is updated. Any distributed trust model can be applied to our proposed mechanism. Users can perform concurrent modifications on the shared documents as well as they can share documents with different specified obligations according to their preferences.

We are currently working on a mechanism that detects cheaters who modified the log for hiding their misbehavior. A direction of future work is the evaluation of the log-auditing mechanism proposed in this paper. We will test first the efficiency and complexity of our algorithms in P2P simulators such as PeerSim[3]. We plan afterward to apply our trust management approach to existing research on P2P online social networks such as PeerSoN [1].

Acknowledgment

This work is partially funded by the ANR national research grant STREAMS (ANR-10-SEGI-010).

[3]http://peersim.sourceforge.net/

REFERENCES

[1] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta. PeerSoN: P2P Social Networking – Early Experiences and Insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems, SNS 2009*, pages 46–52, Nürnberg, Germany, March 2009. ACM Press.

[2] J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini. Audit-based Compliance Control. *International Journal of Information Security*, 6(2):133–151, March 2007.

[3] M. Hilty, D. A. Basin, and A. Pretschner. On Obligations. In *Proceedings of the 10th European Symposium on Research in Computer Security, ESORICS 2005*, pages 98–117, Milan, Italy, September 2005. Springer.

[4] M. Jawad, P. Serrano-Alvarado, and P. Valduriez. Protecting Data Privacy in Structured P2P Networks. In *Proceedings of the 2nd International Conference on Data Management in Grid and Peer-to-Peer Systems, Globe 2009*, pages 85–98, Linz, Austria, 2009. Springer-Verlag.

[5] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web, WWW 2003*, pages 640–651, Budapest, Hungary, May 2003. ACM Press.

[6] K. Krukow, M. Nielsen, and V. Sassone. A Logical Framework for History-based Access Control and Reputation Systems. *Journal of Computer Security*, 16(1):63–101, January 2008.

[7] L. Mui, M. Mohtashemi, and A. Halberstadt. A Computational Model of Trust and Reputation. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, HICSS 2002*, pages 2431–2439, Waikoloa, Big Island, Hawaii, January 2002. IEEE Computer Society.

[8] J. Park and R. Sandhu. The UCONABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7(1):128–174, February 2004.

[9] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles - SOSP'97*, pages 288–301, Saint Malo, France, September 1997. ACM Press.

[10] R. Rodrigues and P. Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, October 2010.

[11] M. Roger and J. Goubault-Larrecq. Log Auditing through Model-Checking. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations, CSFW 2001*, pages 220–234, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[12] R. Sherwood, S. Lee, and B. Bhattacharjee. Cooperative Peer Groups in NICE. *Computer Networks*, 50(4):523–544, March 2006.

[13] G. Stevens and V. Wulf. A New Dimension in Access Control: Studying Maintenance Engineering Across Organizational Boundaries. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW 2002*, pages 196–205, New Orleans, Louisiana, USA, November 2002. ACM Press.

[14] S. Weiss, P. Urso, and P. Molli. Logoot-Undo: Distributed Collaborative Editing System on P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1162–1174, August 2010.

[15] D. I. Wolinsky, P. S. Juste, P. O. Boykin, and R. Figueiredo. OverSoc: Social Profile Based Overlays. In *Proceedings of the Workshop on Collaborative Peer-to-Peer Systems, COPS 2010*, pages 205–210, Larissa, Greece, June 2010. IEEE Computer Society.