Privacy-aware Collaborative Access Control in Web-based Social Networks

Barbara Carminati and Elena Ferrari

Department of Computer Science and Communication
University of Insubria
22100 Varese, (Italy)
{barbara.carminati,elena.ferrari}@uninsubria.it

Abstract. Access control over resources shared by social network users is today receiving growing attention due to the widespread use of social networks not only for recreational but also for business purposes. In a social network, access control is mainly regulated by the relationships established by social network users. An important issue is therefore to devise *privacy-aware* access control mechanisms able to perform a controlled sharing of resources by, at the same time, satisfying privacy requirements of social network users wrt their relationships. In this paper, we propose a solution to this problem, which enforces access control through a collaboration of selected nodes in the network. The use of cryptographic and digital signature techniques ensures that relationship privacy is guaranteed during the collaborative process. In the paper, besides giving the protocols to enforce collaborative access control we discuss their robustness against the main security threats.

Keywords Privacy-preserving data management, Web-based Social Networks, Collaborative access control.

1 Introduction

The last few years have witnessed the explosion of Web-based Social Network (WBSN) users [1]. WBSNs make available an information space where each social network participant can publish and share information, such as personal data, annotations, blogs, and, generically, resources, for a variety of purposes. In some social networks, users can specify how much they trust other users, by assigning them a trust level. Information sharing is based on the establishment of relationships of different types among participants (e.g., colleague of, friend of). However, the availability of this huge amount of information within a WBSN obviously raises privacy and confidentiality issues. For instance, in 2006, Facebook receives the complaints of some privacy activists against the use of the News Feed feature [2], introduced to inform users with the latest personal information related to their online friends. These complaints result in an online petition, signed by over 700,000 users, demanding the company to stop this service. Facebook replayed by allowing users to set some privacy preferences. More recently, November 2007, Facebook receives other complaints related to the use of Beacon [3]. Beacon is part of the Facebook advertising system, introduced to track users activities on more than 40 web sites of Facebook partners. Such information is collected even when the users are off from the social-networking site, and is reported to users friends without the consent of the user itself. Even in this case, the network community promptly reacts with another online petition that gained more than 50,000 signatures in less than 10 days. These are only few examples of privacy concerns related to WBSNs. All these events have animated several online discussions about security and privacy in social networking, and government organizations started to seriously consider this issue [4–7].

To partially answer the security and privacy concerns of their users, recently some social networks, e.g., Facebook (http://www.facebook.com) and Videntity (http://videntity.org), have started to enforce quite simple protection mechanisms, according to which users can decide whether their data, relationships, and, generically, resources, should be public or accessible only by themselves and/or by users with whom they have a direct relationship. However, such mechanisms are not enough, in that they enforce too restrictive and/or too simple protection policies. There is, then, the need of enforcing more flexible strategies, making a user able to define his/her own rules, denoting the set of network participants authorized to access his/her resources and personal information, even though they are not directly connected through a relationship. Additionally, since WBSN information sharing is mostly based on the relationships existing among network participants, there is the need of protecting relationship information when performing access control. For instance, a user would like to keep private the fact that he/she has a relationship of a given type with a certain user. Therefore, in [8] we have proposed a framework to enforce client-side access control to WBSN resources, according to which the requestor must provide the resource owner with a *proof* of being authorized to access the requested resource. The proposed access control is *privacy-aware* in the sense that privacy requirements referring to relationships established by WBSN users are preserved when enforcing access control. Access control requirements are expressed in terms of relationship types, depths, and trust levels. In [8], relationships are encoded through certificates and their protection requirements are expressed through a set of *distribution rules*, which basically state who can exploit a certificate for access control purposes. Relationship privacy is enforced by encrypting a certificate with a symmetric key which is delivered only to users satisfying the corresponding distribution rule. Encrypted certificates are stored at a central node, which does not receive the corresponding decryption key, and therefore it is not required to be trusted. However, the mechanism proposed in [8] has the following shortcomings:

- It relies on a central node to store relationship certificates that may become a bottleneck and may be vulnerable to DoS attacks;
- The central node must be trusted wrt certificate revocation enforcement when a relationship does not exist anymore, in that, according to the architecture proposed in [8], the central node maintains a certificate revocation list which must be updated to reflect social network topology changes;

- As pointed out in [8], one of the drawback of client-side access control is that revealing access rules regulating access to the requested resource may compromise the privacy of resource owner relationships. The reason is that, when a user requests a resource, the resource owner replies with an access rule, which contains, among other information, the relationships in which the requestor must be involved in order to gain the access. If the owner wants to keep private some types of relationships he/she has with other network nodes, this mechanism could lead to privacy breaches.

To overcome these shortcomings, in this paper, we propose an alternative way of enforcing access control wrt the solution presented in [8]. Rather than using the client-side paradigm, access control is enforced through the collaboration of selected nodes in the network. The collaboration is started by the resource owner, on the basis of the access rules regulating the access to the requested resource. The owner contacts only the nodes that satisfy its distribution rules, thus avoiding the privacy breaches related to access rules distribution discussed above. The aim of the collaboration is to provide the owner with a *path*, proving that the requestor has the relationships required to gain access to the resource. Since each node taking part in the collaboration is aware of the relationships existing among the other nodes taking part in the process, the collaborative process is driven by the specified distribution rules: a node is invited to collaborate only if it satisfies the distribution rules of the other nodes taking part in the collaboration. Encryption and signature techniques are used to avoid trust levels disclosure and forgery, as well as to make a node able to verify the correct enforcement of distribution rules.

WBSN privacy and security is a new research area. Up to now, research has mainly focused on privacy-preserving techniques to mine social network data. The main goal of this research is to avoid as much as possible the disclosure of private information about WBSN members when analyzing WBSN data for statistical purposes [9–12]. In contrast, in the field of access control (apart from [8, 14]) very little work has been so far reported. The only other proposal we are aware of is the one discussed in the position paper by Hart, Johnson, and Stent [13]. The access control model presented in [13] uses existing WBSN relationships to denote authorized members, however only the direct relationships they participate in are considered, and the notion of trust level is not used in access authorizations. Resources are not denoted by their identifiers, but based on their content. Information about resources' content is derived based on users' tags and by content analysis techniques. However, [13] does not provide any information about access control enforcement, nor they consider relationship privacy protection when enforcing access control, which is the focus of the current paper.

This paper builds on some previous work we have done in the field of privacy-aware access control in WBSNs. In particular, [14] presents the access control model on which the enforcement mechanism described in this paper relies. [8] adds to the model proposed in [14] the possibility of expressing privacy requirements on relationships established by WBSN users. However, in [8] access control enforcement relies on a central node storing encrypted certificates to be used

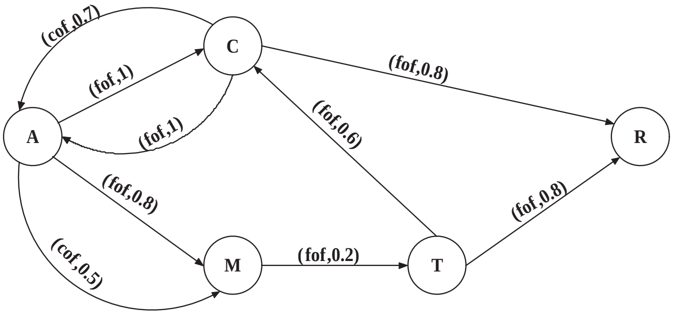

Fig. 1. A portion of a WBSN

for gaining access to a resource. In the current paper, we propose an alternative enforcement mechanism, wrt the one presented in [8], where access control enforcement is obtained through a collaboration among selected nodes in the network.

The remainder of this paper is organized as follows. Next section introduces some preliminary concepts needed in the remainder of the paper. Section 3 presents our collaborative access control protocol, whereas Section 4 analyzes its security. Finally, Section 5 concludes the paper and outlines directions for future work.

2 Preliminary concepts

In what follows, we model a WBSN $\mathcal{SN}$ as a tuple $(V_{\mathcal{SN}}, E_{\mathcal{SN}}, RT_{\mathcal{SN}}, T_{\mathcal{SN}}, \phi_{E_{\mathcal{SN}}})$, where $V_{\mathcal{SN}}$ and $E_{\mathcal{SN}}$ are, respectively, the nodes and edges of a graph, $RT_{\mathcal{SN}}$ is the set of supported relationship types, $T_{\mathcal{SN}}$ is the set of supported trust levels, and $\phi_{E_{\mathcal{SN}}}\colon E_{\mathcal{SN}} \rightarrow RT_{\mathcal{SN}} \times T_{\mathcal{SN}}$ is a function assigning to each edge $e \in E_{\mathcal{SN}}$ a relationship type $rt \in RT_{\mathcal{SN}}$ and a trust level $t \in T_{\mathcal{SN}}$. The number and type of relationships in $RT_{\mathcal{SN}}$ and trust levels in $T_{\mathcal{SN}}$ depend on the specific social network and its purposes. Similarly, each WBSN supports a different range and type of trust levels, corresponding either to a set of integers or rational numbers, or to Boolean values. In case a WBSN does not support trust, this means that all the nodes are equally trustworthy, and thus we assume that with each edge is associated the maximum level of trust. Given an in/direct relationship of type rel between nodes v and $v^{'}$, the trust level $t_{vv^{'}}^{rel}$ denotes how much v trusts $v^{'}$ wrt relationship rel. We also assume the existence of a central node $\mathcal{CN}$ that is in charge of user registration and management.

Example 1. A simple example of WBSN is depicted in Figure 1. In the figure, the initial node of an edge is the node which established the corresponding relationship. Labels associated with edges denote, respectively, the type and trust level of the corresponding relationship. With reference to Figure 1, `A(lex)` is friend-of (fof) both `C(arl)` and `M(ark)`. However, `A` trusts `C` more than `M`. `A` and `T(ed)` are indirect friends due to the fof relationships existing between `A` and M and M and T. C is also a colleague-of (cof) A, and the trust assigned to this relationship by C is 0.7.

Following the model proposed in [14], we assume that each resource rsc to be shared in the $\mathcal{SN}$ is protected by a set of *access rules*. Each access rule has the form (rid, AC), where AC is a set of *access conditions* that need to be all satisfied in order to get access to rid. An access condition is a tuple $(v, rt, d_{max}, t_{min})$, where v is the node with whom the requestor must have a relationship of type rt, whereas d_{max} and t_{min} are, respectively, the maximum depth, and minimum trust level that the relationship should have. The depth of a relationship of type rt between two nodes v and $v^{'}$ corresponds to the length of the shortest path between v and $v^{'}$ consisting only of edges labeled with rt. For instance, with reference to Figure 1, the depth of the fof relationship between A and R(obert) is 2. In contrast, the trust level associated with a path is computed by multiply all the trust levels associated with the edges in the path, even if other formulas to compute the trust level can be used as well. In this paper, we constrain the access conditions contained into an access rule by assuming that v can be only equal to the owner of the resource to be protected. As it will be clarified in what follows, this assumption makes the resource owner able to start the collaborative process needed to answer an access request. Additionally, this is not a too strong restriction because it is very common that most of the access control requirements in a $\mathcal{SN}$ are expressed by an owner in terms of the relationships it holds with other nodes in the network, rather than in terms of relationships in which it is not involved.

Like in [8], relationship privacy requirements are stated through *distribution rules*. A distribution rule on a relationship of type rt established by node A with node B, denoted in what follows as $DR^{rt}_{\mathtt{AB}}$, is a triple of the form $(v, \overline{rt}, d_{max})$, stating that the only nodes authorized to be aware of the relationship of type rt established by A with B are those that have a relationship of type $\overline{rt}$ with v and maximum depth d_{max}. Similarly to access rules, in this paper we constrain distribution rules specification by assuming that v can only be A and that $\overline{rt}$ is fixed to the type of the relationship to which the distribution rule applies, that is, rt. For instance, given a relationship of type fof between A and B, the corresponding distribution rule $DR^{fof}_{\mathtt{AB}}$ should have the node component equal to A and the relationship type component equal to fof. Once again this restriction makes a node able to correctly enforce distribution rules when performing collaborative access control (see Section 3 for the details).

Example 2. Consider the WBSN in Figure 1. Suppose that doc_1 is a resource owned by A. Suppose that A wishes to make doc_1 accessible only to its direct or indirect friends of maximum depth three, with the constraint that their trust level must be at least equal to 0.8. These requirements can be expressed by the following access rule: $\mathtt{AR}_1 = (\mathtt{doc}_1, \{(\mathtt{A,fof,3,0.8})\})$. In contrast, suppose that A specifies the following distribution rule $\mathtt{DR}^{fof}_{\mathtt{AM}} = (\mathtt{A,fof,3})$, this means that the relationship of type fof existing between A and M can be disclosed only to friends of A with maximum depth three, that is, M, T, R, and C. Finally, if M specifies the following distribution rule: $\mathtt{DR}_{\mathtt{MT}}^{fof} = \mathtt{(M,fof,2)}$, it means that the fof relationship existing between M and T can be seen only by M's direct friends and by the direct friends of M's direct friends, that is, T, R, and C.

We assume that each node n_i in a WBSN owns a key pair (SK_i, PK_i), where SK_i is the private key, whereas PK_i is the corresponding public key. We denote with $SK_i(x)$ (resp. $PK_i(x)$) the encryption of x with key SK_i (resp. PK_i). We further assume PK_i to be known by all the nodes wishing to getting in touch with n_i. Moreover, we assume that each time a node n_1 wishes to establish a relationship with another node n_2, it informs n_2 about that. Therefore, the corresponding relationship certificate is signed by both n_1 and n_2. This means that each node is aware of the relationships established with it by other nodes. We think that it is very important that a node is aware of the in-coming edges, since relationships are the basis of access control enforcement. Otherwise, a node can cheat and claim that it has a relation with others even if this is not true. For instance, if Bob is a close friend of the boss of a big company, Ann can establish a relation of type fof with him just to become an indirect friend of the boss. If Bob is not informed about that, there is no way of preventing this behaviour. However, the nodes are not aware of the trust levels of in-coming relationships, because we believe that this is a confidential information that shall be kept private (that is, a node n_1 might not want to reveal to another node n_2 how much it trusts it).

3 Collaborative access control

In this section, we illustrate our proposal for enforcing access control while preserving WBSN relationship privacy. We start by providing an overview of the approach, then we present the related protocols and an example of their execution.

3.1 Overview of the approach

Differently from [8], access control is enforced through a *collaboration* among selected nodes in the $\mathcal{SN}$. The collaboration is needed to prove to the owner that the node requesting a resource satisfies the requirements (in terms of relationships it holds and corresponding trust levels and relationship depths) stated by the owner access rules. If the result of the collaboration is the identification of a path with the requirements stated by the owner access rules, then the access is granted. Otherwise, it is denied. The collaborative process is started by the owner, on the basis of the access rules regulating the access to the requested resource. This avoids the possible privacy breaches pointed out in the introduction due to distribution of access rules. In particular, the owner starts the collaboration by sending a request to its direct neighbours. More precisely, it contacts all the neighbours with which it has established a relationship of the type required by the access control rule associated with the requested resource, asking whether they have a relationship of the required type with the requestor node. If a resource is protected by more than one access rule, the process is iterated till the access can be granted or till all the access rules have been considered. For simplicity, in the following, we assume access rules consisting only of one access condition. The protocols can be easily extended by iterating the described operations for all the access conditions contained in an access rule. Once a node different from the requestor receives a request for collaboration, it propagates the request to those of its neighbours with which it has established a relationship of the required type. This process is iterated until a node having a relationship of the required type with the requestor is reached, or until the request can no longer be propagated along the network.

To verify whether an access can be granted or not the owner must be provided not only with a path of the required type and depth, but it also must know the trust level of all the edges in the returned path. Therefore, when propagating the request for collaboration, a node forwards also the trust level of its relationship. To avoid that intermediate nodes are aware of the trust levels assigned by the nodes in the path, the trust level is encrypted with the owner's public key. Moreover, we also need a mechanism to avoid that a node can repudiate the trust level it has inserted and, therefore, to minimize as much as possible the insertion of fake trust levels. To this purpose, each node signs the encryption of the trust level it has inserted.

Since each node adds its identity to the path it receives before sending it to the subsequent node, each node in the path from the owner to the requestor is aware of the relationships existing among the nodes in the path, and this could not respect the privacy requirements that a node might have on its relationships. For this reason, the collaboration process is driven by the specified distribution rules. More precisely, a node is required by another node to participate in a collaboration only if it satisfies the distribution rules associated with *all* the relationships in the path built so far. Thus, before propagating the request for collaboration to a neighbour node, a node must verify whether the neighbour satisfies all the distribution rules associated with the relationships in the path built so far. Introducing distribution rules enforcement in collaborative access control requires to devise a mechanism avoiding untrusted nodes to require collaboration of neighbour nodes even if they do not satisfy distribution rules in the path, or at least making it possible to detect misbehaviours. To this purpose, we use signature techniques. In particular, a node attaches to the request for collaboration sent to one of its neighbours, the distribution rule for the relationship it discloses and its signature over it. Moreover, it digitally signs all the signed distribution rules contained in the received message, referring to previously disclosed relationships. As it will be clear in the following section, this sort of "*onion signature*" makes a node able to detect if all the previous nodes in the path have correctly enforced the distribution rules.

3.2 Access control protocol

As stated in the previous section, access control enforcement is obtained through a collaboration among nodes in the $\mathcal{SN}$. The collaboration has the aim of identifying a *path* in the $\mathcal{SN}$ satisfying the requirements stated by the access rules specified for the requested resource. The notion of path is formalized as follows.

Definition 1 (Relationship path). *Let* $\mathcal{SN} = (V_{\mathcal{SN}}, E_{\mathcal{SN}}, RT_{\mathcal{SN}}, T_{\mathcal{SN}}, \Phi_{E_{\mathcal{SN}}})$ *be a WBSN. A relationship path in* $\mathcal{SN}$ *is a pair* $(rt, node\_list)$*, where* $rt \in RT_{\mathcal{SN}}$ *is a relationship type and node_list is an ordered list* $\langle n_1, \ldots, n_k\rangle$*,* $n_1, \ldots, n_k \in V_{\mathcal{SN}}$ *of* $\mathcal{SN}$ *nodes such that for each pair* $n_i, n_{i+1} \in node\_list$*,* $i = 1, \ldots, k-1$*, there exists an edge* $(n_i, n_{i+1}) \in E_{\mathcal{SN}}$ *labeled with relationship type rt.*

In what follows, we use the dot notation to refer to specific components within a tuple.

Since the nodes to be contacted are selected on the basis of the distribution rules defined for the relationships in the path built so far, each node receiving a partial path must be aware of the corresponding distribution rules. Moreover, each node receiving a request to collaborate must be able to verify whether previous nodes in the path have correctly enforced the distribution rules for all the relationships in the path. To this purpose, each node that takes part in the collaborative process inserts in the message to be sent to the subsequent node the distribution rule associated with the relationship it inserts into the relationship path, as well as its signature. Additionally, it signs all the signatures of the distribution rules contained in the message it receives, if it verifies that they all have been correctly enforced by previous nodes. Distribution rules and corresponding signatures for the relationships in a path are stored into a data structure called *Onion signature*, defined next. In what follows, we denote with $Sign_i(x)$ the signature of node n_i over x.

Definition 2 (Onion signature). *Let* $p = (rt, node\_list)$ *be a relationship path for a social network* $\mathcal{SN}$*, where* $node\_list = \langle n_1, \ldots, n_k\rangle$*. The Onion signature data structure for path p, denoted as Onion_signature(p), is an ordered list of pair* $(DR^{rt}_{n_i n_{i+1}}, Signature_i)$*,* $i = 1, \ldots, k-1$ *where* $DR^{rt}_{n_i n_{i+1}}$ *is the distribution rule specified by node* n_i *for the relationship of type rt connecting* n_i *to* n_{i+1}*, whereas* $Signature_i = Sign_k(Sign_{k-1}(\ldots Sign_i(DR^{rt}_{n_i n_{i+1}})))$*.*

Example 3. Consider the WBSN of Figure 1 and the distribution rules of Example 2. The following is an example of relationship path: $(\mathtt{fof}, \langle \mathtt{A}, \mathtt{M}, \mathtt{T}\rangle)$ stating that there is an indirect fof relationship between A and T. According to the distribution rules introduced in Example 2. The corresponding onion signature is: $\langle((\mathtt{A}, \mathtt{fof}, 3), \mathtt{Sign_T}(\mathtt{Sign_M}(\mathtt{Sign_A}(\mathtt{A}, \mathtt{fof}, 3)))), ((\mathtt{M}, \mathtt{fof}, 2), \mathtt{Sign_T}(\mathtt{Sign_M}(\mathtt{M}, \mathtt{fof}, 2)))\rangle$.

Enforcement of collaborative access control is performed by Algorithm 1. The resource owner, upon receiving an access request, retrieves from its Policy Base the access rules regulating the access to the requested resource (step 1). For simplicity, in the algorithm, we assume a single access rule consisting only of one access condition. The algorithm can be easily extended to more access

Algorithm 1 The collaborative access control protocol

```
INPUT: An access request $(req, res)$ submitted to node $own$ by $req$
OUTPUT: $res$, if $req$ satisfies the access control requirements of $own$,
        an access denied message otherwise.
1. $own$ retrieves from its Policy Base the access rule $AR$ associated with resource $res$
2. Let $AC = (own, rt, d, t)$ be the access condition in $AR$
3. Let $C\_nodes$ be the set of nodes with which $own$ established a relationship of type $AC.rt$
4. Foreach $n \in C\_nodes$:
     (a) $p := (rt, \langle own \rangle)$
     (b) $msg\_res := SendCollReq(\{req, own, p, \langle (DR^{rt}_{ownn}, Sign_{own}(DR^{rt}_{ownn})) \rangle\}, n)$
     (c) $i := msg\_res$
     (d) While $(i \neq \emptyset)$:
        i Let $\overline{msg}$ be the $i$-$th$ received message containing a path
        ii If $check\_DR(\overline{msg}.path, \overline{msg}.onion\_sign) = \emptyset$:
             1. Let $depth$ be the depth of $\overline{msg}.path$
             2. Let $trust$ be the trust computed by using the trust values in $\overline{msg}$
             3. If $trust \geq AC.t$ and $depth \leq AC.d$: Return $res$
        EndIf
        iii $i := i$ -1
     EndWhile
endfor
5. Return $access\ denied$
```

rules, each one consisting of more than one access condition, by simply iterating the steps we are describing in what follows. Then, the owner identifies the set of nodes with which it has established a relationship of the type *rt* required by the access condition contained in the considered access rule (step 3). It iteratively considers (step 4) each node in this set and sends it a message to start the collaboration process. The message, sent by function *SendCollReq()* in step 4.b, contains the owner and requestor identifiers, the distribution rule associated with the relationship of type *rt* existing between the owner and the receiving node, the signature of the distribution rule generated by the resource owner, and the path built so far (which consists only of the owner itself).

Once the message has been sent, the algorithm waits for the node reply, which consists of a null value, if no path satisfying the stated confidentiality and privacy requirements can be found, or the number of identified paths, otherwise. In case *SendCollReq()* returns a not null value, a message containing each of the identified paths is sent to the owner (see procedure *path_builder()* in Figure 2 explained next). The message contains information on the identified paths (e.g., the nodes composing it, the corresponding trust levels and the onion signature). The algorithm first verifies whether all the nodes in the received path have correctly enforced the signed distribution rules contained in the message (step 4.d.ii). This is done by function *check_DR()*, presented in Figure 3. If the check succeeds, the algorithm computes the depth of the received path and its trust level and, if they both satisfy the constraints stated in the access rule, the access is granted (step 4.d.ii.3). Otherwise the process is iterated on the next received message, until there are no more message to be processed. Then, if the access has not been granted, the collaboration is requested to the next node in the set identified in step 3, until the access is granted or all the nodes in the set have been contacted, without finding a suitable path. In this case, the access is denied (step 5).

```
Procedure $path\_builder(n, msg)$

1. Let $sender$ be the node from which message $msg$ has been received
2. Let $Distr\_rules$ be the set of distribution rules contained in $msg$
3. If $check\_DR(msg.path, msg.onion\_sign) = \emptyset$:
    (a) Let $TOT\_msg\_res$ be initialized to 0
    (b) Let $C\_nodes$ be the set of nodes with which $n$ has established
    a relationship of type $msg.path.rt$
    (c) If $C\_nodes = \emptyset$:
          Send $TOT\_msg\_res$ to $sender$
          Return
    endif
    (d) Foreach $\overline{n} \in C\_nodes$:
            i If $\overline{n}$ satisfies all rules in $Distr\_rules$:
                1. Let $\overline{msg}$ be a copy of $msg$
                2. Update $path$ in $\overline{msg}$ by adding $n$
                3. Add $PK_{own}(t_{n\overline{n}}), Sign(PK_{own}(t_{n\overline{n}}))$ to $\overline{msg}$
                4. Replace the onion signature in $\overline{msg}$ with $Onion\_signature(\overline{msg}.path)$
                5. If $\overline{n} = msg.req$:
                    (a) If $msg.own$ satisfies all rules in $Distr\_rules \cup DR_{n\overline{n}}^{msg.path.rt}$:
                          i Send $\overline{msg}$ to $msg.own$
                          ii $TOT\_msg\_res := TOT\_msg\_res + 1$
                    endif
                else
                    $msg\_res := SendCollReq(\overline{msg}, \overline{n})$
                    $TOT\_msg\_res := TOT\_msg\_res + msg\_res$
            endif
    endfor
    (e) Send $TOT\_msg\_res$ to $sender$
else:
    (f) Send $msg\_res = \emptyset$ to $sender$
    (g) Send$(n, msg, check\_DR(msg.path, msg.onion\_sign))$ to $msg.own$ and $\mathcal{CN}$
endif
```

Fig. 2. Procedure *path_builder()*

Each time a node n receives a request for collaboration, it executes procedure *path_builder()*, presented in Figure 2. The procedure processes the received message and decides the next action to be performed. In particular, it initializes variable TOT_msg_res to zero (step 3.a). This variable is used to store the number of identified paths. Then, the procedure identifies the nodes with which n has established a relation of the type rt of the relationship path contained into the received message (step 3.b). If this set is empty, it halts by returning TOT_msg_res to the sender, since no other path can be found (step 3.c). Otherwise, for each node $\overline{n}$ in the computed set, it verifies whether $\overline{n}$ satisfies all the distribution rules contained into the received message (step 3.d.i). In this case, n updates the received message, by adding itself to the path, and the encrypted and signed trust level of the relationship between n and $\overline{n}$. It also updates accordingly the onion signature contained into the received message. Then, it verifies whether $\overline{n}$ is the requestor node (step 3.d.i.5). If this is the case, n verifies whether the owner satisfies all the distribution rules in the received message plus the distribution rule regulating the disclosure of the relationship existing between n and $\overline{n}$.

If this is the case, it sends the updated message to the owner and updates accordingly variable TOT_msg_res. In contrast, if $\overline{n}$ is not the requestor node, n sends it the collaboration request and updates accordingly variable TOT_msg_res. When all the possible requests of collaboration have been sent and the corresponding replies received, n sends TOT_msg_res to the sender (step 3.e).

Before processing the received message, *path_builder()* verifies whether all the nodes in the path received as input have correctly enforced the signed distribution rules contained into the received message (step 3). In case this check fails, n notifies the owner and the $\mathcal{CN}$ that one or more nodes have not correctly enforced the specified distribution rules contained in the received message (step 3.g). This information can be used to perform subsequent actions against the malicious nodes (e.g, notification to other nodes of their incorrect behaviours, temporary banning from the WBSN and so on).

Checking the correct enforcement of distributions rule is done by function *check_DR()* illustrated in Figure 3. *check_DR()* takes as input the path contained in the message and the corresponding onion signature and returns the set of nodes, if any, which did not correctly enforce the distribution rules. If the returned set is empty it means that all the nodes in the path have correctly enforced the corresponding distribution rules.

3.3 An illustrative example

Consider the $\mathcal{SN}$ shown in Figure 1 and the access and distribution rules of Example 2 and suppose that `R` requires doc_1 to `A`. According to the protocols described in Section 3, `A` first of all retrieves from his Policy Base the access rules regulating the access to doc_1, i.e., $\texttt{AR}_1 = \texttt{(doc}_1\texttt{,\{(A,fof,3,0.8)\})}$. According to Algorithm 1, `A` contacts his fof neighbours. Let us suppose it starts to contact node `M` by sending it a message containing the following components:

- path $p = (fof, \langle \texttt{A} \rangle)$;
- $req = \texttt{R}$, $own = \texttt{A}$;

```
Function $check\_DR(path, onion\_sign)$

1. $Bad\_nodes := \emptyset$
2. Let $l$ be the length of $path.node\_list$
3. For $i = 1$ to $l$:
        (a)$flag := 0$; $k := 0$
        (b) Let $DR$ be the distribution rule in $onion\_sign[i]$
        (c) Let $sign$ be the signature in $onion\_sign[i]$
        (d) For $j = l$ to $i$:
                i If $validate\_sign(node\_list[j], sign) = 0$:
                    1. Add $node\_list[j]$ to $Bad\_nodes$
                    2. $flag$ :=1
                    3. Break
                endif
                ii $k := k + 1$
        endfor
        (e) If $flag \neq 1$:
              i If $k > DR.d_{max}$ +1:
                    1. For $y = i + DR.d_{max}$ to $l$: Add $node\_list[y]$ to $Bad\_nodes$
endfor
4. Return $Bad\_nodes$
```

Fig. 3. Function *check_DR()*

- onion_sign $= \langle((\mathtt{A,fof,3}), Sign_{\mathtt{A}}((\mathtt{A,fof,3})))\rangle$, where $(\mathtt{A,fof,3}) = DR_{\mathtt{AM}}^{fof}$.

Once M receives the request for collaboration message, it runs procedure *path_builder* (cfr. Figure 2). The procedure first checks the correct enforcement of the distribution rules in the path, by verifying the onion signature data structure through function *check_DR()* (see Figure 3); then, it considers the only node with which M has a relationship of type fof, that is, T. It verifies whether T satisfies the distribution rules in the received message, the only one is (A,fof,3) which is satisfied by T. Therefore, it adds itself to the path and it inserts $PK_{\mathtt{A}}(\mathtt{t}_{\mathtt{MT}}^{fof})$ in the message, that is, the encryption of the trust level M has assigned to T with the public key of the resource owner. Moreover, it updates the onion signature data structure as follows: $\langle((\mathtt{A,fof,3}), Sign_{\mathtt{M}}(Sign_{\mathtt{A}}((\mathtt{A,fof,3})))),((\mathtt{M,fof,2}), Sign_{\mathtt{M}}((\mathtt{M,fof,2})))\rangle$, where $(\mathtt{M,fof,2}) = DR_{\mathtt{MT}}^{fof}$. Then, since T is not the requestor node, M sends the request for collaboration to T. T first of all verifies the received onion signature. Then, it determines the set of its fof neighbours, that is, $\{\mathtt{C,R}\}$. It selects one of the node in this set, suppose R, and verifies whether it satisfies all the distribution rules contained into the received message. Since the check succeeds, it updates the path by inserting itself, it updates the received message by adding $PK_{\mathtt{A}}(\mathtt{t}_{\mathtt{TR}}^{\mathtt{fof}})$ and it updates the onion signature data structure by adding and signing $DR_{\mathtt{TR}}^{fof}$. Moreover, it signs all the distribution rules contained into the received onion signature. Since R is the requestor node, T verifies whether A satisfies all the distribution rules contained into the received message plus $DR^{fof}_{\texttt{TR}}$. In this case, A does not satisfy $DR^{fof}_{\texttt{MT}}$ = (M,fof,2),[1] therefore T tries to build a valid path by contacting its other fof neighbour, that is, C. However, when C receives the message, it verifies that R does not satisfy one of the distribution rules in the received message, that is, $DR^{fof}_{\texttt{MT}}$, thus it sends a null message to T.[2] Since T does not have any other node to contact, it sends a null message back to M, which in turn sends a null message back to the owner A. Therefore, A contacts its other friend, that is, C by sending it a message similar to the one sent to M and consisting of the following components:

- path $p = (fof, \langle \texttt{A} \rangle)$;
- req = R, own = A;
- onion_sign = $\langle ((\texttt{A,fof,4}), Sign_{\texttt{A}}((\texttt{A,fof,4}))) \rangle$, where (A,fof,4) = $DR^{fof}_{\texttt{AC}}$;[3]

Once C receives the request for collaboration, it first checks the correct enforcement of the distribution rules in the path, then it verifies whether R satisfies $DR^{fof}_{\texttt{AC}}$. Since this is the case, it adds itself to the received path and it inserts in the message $PK_{\texttt{A}}(\texttt{t}^{\texttt{fof}}_{\texttt{CR}})$, that is, the encryption of the trust level C has assigned to R with the public key of the resource owner. Moreover, it updates the onion signature data structure as follows: $\langle ((\texttt{A,fof,4}), Sign_{\texttt{C}}(Sign_{\texttt{A}}((\texttt{A,fof,4})))), ((\texttt{C,fof,3}), Sign_{\texttt{C}}(\texttt{C,fof,3})) \rangle$, where (C,fof,3) = $DR^{fof}_{\texttt{CR}}$. Since R is the requestor node, C verifies whether A satisfies all the distribution rules in the updated message. Since the check succeeds, it sends the updated message to A, as well as a message containing the value 1. Upon receiving the 1 message, A processes the other message received by C. It first verifies the correctness of the contained onion signature data structure, then it computes the trust level of R on the basis of the trust levels contained into the received message. According to Figure 1, $\texttt{t}^{fof}_{\texttt{AR}} = 1 * 0.8 = 0.8$. The length of the received path is 2. Thus, $\texttt{AR}_1$ = (doc$_1$,{(A,fof,3,0.8)})is satisfied and therefore R can access doc_1.

4 Security Analysis

In this section, we discuss the robustness of our system against possible attacks. As adversary model, we assume that the adversary is a node in the $\mathcal{SN}$ which can collude with other network nodes to attack the system. To keep the discussion simple, in this paper we have not complemented the proposed protocols with techniques generally used to protect data during transmission. However, we

[1] According to the semantics of the distribution rules introduced in [8], A satisfies $DR^{fof}_{\texttt{MT}}$ = (M,fof,2) if there exists a *fof* path of length not greater than 2, having M as source node and A as terminal node.

[2] Note that, C may not be aware of the fof relationship between T and R, and therefore, by considering only the information in the received message, it deduces that $DR^{fof}_{\texttt{MT}}$ is not satisfied by R.

[3] In this example, we assume that the distribution rule specified by A for the fof relationship with C is $DR^{fof}_{\texttt{AC}}$ = (A,fof,4), whereas the distribution rule specified by C for the fof relationship with R is $DR^{fof}_{\texttt{CR}}$ = (C,fof,3).

are aware that the current version of the proposed protocols can be subject to eavesdropping and replay attacks. These kinds of attacks can be easily avoided adopting well-know mechanisms [15]. Therefore, the main attacks that a node can perform during collaborative access control are the following:

- Learn the trust level of previous nodes in the path.
- Alter the received trust level or insert a fake one.
- Incorrectly enforce distribution rules.

Let us consider all the above three attacks in details. Trust levels forwarded among nodes are encrypted with the public key of the resource owner. As such, provided that the computational power of a node does not make it able to break the cryptosystem, each intermediate node in the path is not able to gain access to the trust levels. The fact that each node in the path signs the trust level it sends to the next node provides non-repudiation of the inserted trust levels. Additionally, if a node alters the trust level inserted by another node it can be detected by either the owner or any node in the path, since this alteration invalidates the node's signature.

Finally, the onion signature data structure makes a node able to verify whether the previous nodes in the path have correctly enforced the distribution rules. This is performed by function *check_DR*() by performing two different checks for each distribution rule `DR` (see Figure 3 for the details). First, the function verifies whether all the nodes have properly signed `DR` (step 3.d.i). This makes sure the owner that all nodes are aware of distribution rule `DR`. If the signature of a node n fails to be validated, *check_DR*() informs the owner and $\mathcal{CN}$ that n did not have correctly performed the protocol. The second check performed by function *check_DR*() over `DR` verifies the constraint on the maximum depth specified in the distribution rule (step 3.e.i). This check makes the owner sure that the distribution rule `DR`, and as a consequence the corresponding relationship, have been disclosed only to nodes whose distance from the node n stating `DR` is less than or equal to the maximum depth stated in `DR`. All the nodes disclosing `DR` and having distance from n greater than the maximum depth in the rule are reported to the owner and $\mathcal{CN}$ as bad nodes.

It is relevant to note that if two or more nodes collude, they can validate the onion signature of previous nodes even if the corresponding distribution rules are not correctly enforced. However, since each node in the path verifies the distribution rules, the set of nodes that have to collude to perform the attack could be very large. Moreover, a final check on the correct distribution rule enforcement is made by the resource owner (Algorithm 1, step 4.d.ii).

5 Conclusions

In this paper we have presented a protocol on support of privacy-aware access control in WBSNs, based on a collaboration of selected nodes in the network. The protocol is based on the use of cryptographic and digital signature techniques, and ensures that relationship privacy is guaranteed during the collaborative process.

We plan to extend the work reported in this paper along several directions. First, an implementation of the collaborative access control protocol is currently under way. In the actual version of the prototype the WBSN nodes are implemented as Web services, whereas the system interface available to users is provided as an extension to the Mozilla Firefox browser. However, we plan to investigate as future work how the API defined by Google OpenSocial initiative [16] can be integrated into the current prototype. The prototype will make us able to test the feasibility of the proposed methods for different social network topologies and application domains. In particular, our techniques are not meant for general-purpose WBSNs (like for instance Facebook or MySpace). Rather, our target scenarios are social networks used at the intranet level or by virtual organizations, that is, geographically distributed organizations whose members are bound by a long-term common interest or goal, and who communicate and coordinate their work through information technology. This is in line with the emerging trend known as Enterprise 2.0 [17], that is, the use of Web 2.0 technologies, like blogs, wikis, and social networking facilities, within the Intranet, to allow for more spontaneous, knowledge-based collaboration. Therefore, we plan to test our prototype implementation in these reference scenarios.

We also plan to investigate how our collaborative access control enforcement can be deployed when access control paradigms different from the one considered in this paper are used (e.g., audit-based access control [18]).

Acknowledgements

The work reported in this paper is partially funded by the European Community under the QUATRO Plus project (SIP-2006- 211001).

References

1. Staab, S., Domingos, P., Mika, P., Golbeck, J., Ding, L., Finin, T.W., Joshi, A., Nowak, A., Vallacher, R.R.: Social networks applied. IEEE Intelligent Systems **20**(1) (2005) 80–93
2. Chen, L.: Facebook's feeds cause privacy concerns. the amherst student. Available at: http://halogen.note.amherst.edu/ astudent/2006-2007/issue02/news/01.html. (October 2006)
3. Berteau, S.: Facebook's misrepresentation of beacon's threat to privacy: Tracking users who opt out or are not logged in. Security Advisor Research Blog. Available at: http://community.ca.com/blogs/securityadvisor/archive/2007/11/29/facebook-s-misrepresentation-of-beacon-s-threat-to-privacy-tracking-users-who-opt-out-or-are-not-logged-in.aspx (2007)
4. Canadian Privacy Commission: Social networking and privacy. Available at: http://www.privcom.gc.ca/information/social/index_e.asp (2007)

5. EPIC: Social networking privacy. Available at: http://epic.org/privacy/socialnet/default.html (2008)
6. Federal Trade Commission: Social networking sites: A parents guide. Available at: http://www.ftc.gov/bcp/edu/pubs/consumer/tech/tec13.shtm (2007)
7. Hogben, G.: Security issues and recommendations for online social networks. Position Paper 1, European Network and Information Security Agency (ENISA). Available at: http://www.enisa.europa.eu/doc/pdf/deliverables/enisa_pp_social_networks.pdf (2007)
8. Carminati, B., Ferrari, E., Perego, A.: Private relationships in social networks. In: ICDE 2007 Workshops Proceedings, IEEE CS Press (2007) 163–171
9. L. Backstrom, C.D., Kleinberg, J.: Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. In: Proceedings of the World Wide Web Conference. (2007)
10. Frikken, K.B., Golle, P.: Private social network analysis: How to assemble pieces of a graph privately. In: Proceedings of the 5th ACM Workshop on Privacy in Electronic Society (WPES 2006). (2006) 89–98
11. Hay, M., Miklau, G., Jensen, D., Weis, P., Srivastava, S.: Anonymizing social networks. Technical Report 07-19, University of Massachusetts Amherst, Computer Science Department (2007)
12. Zheleva, E., Getoor, L.: Preserving the privacy of sensitive relationships in graph data. In: Proceedings of the 1st ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD 2007). (2007)
13. M. Hart, R.J., Stent, A.: More content - less control: access control in the web 2.0. In: Proceedings of the Web 2.0 Security and Privacy Workshop. (2007)
14. Carminati, B., Ferrari, E., Perego, A.: Rule-based access control for social networks. In: Proceedings of the OTM 2006 Workshops. Volume 2 of LNCS 4278, Springer Verlag (2006) 1734–1744
15. Stallings, W.: Network security essentials: applications and standards. Prentice Hall (2000)
16. OpenSocial, G.: Opensocial api v0.7. Available at: http://code.google.com/apis/opensocial/articles/persistence.html
17. McAfee, A.: Enterprise 2.0: The dawn of emergent collaboration. MITSloan Management Review **47**(3) (2006) 21–28
18. Cederquist, J., Corin, R., Dekker, M., Etalle, S., den Hartog, J., Lenzini, G.: Audit-based compliance control. International Journal of Information Security **6**(2-3) (2007) 133–151