Privacy Homomorphisms for Social Networks with Private Relationships

Josep Domingo-Ferrer, Alexandre Viejo, Francesc Sebé, and Úrsula González-Nicolás

Rovira i Virgili University
Department of Computer Engineering and Mathematics
UNESCO Chair in Data Privacy
Av. Països Catalans 26, Tarragona, Catalonia.
E-mail: {josep.domingo, alexandre.viejo, francesc.sebe, ursula.gonzaleznicolas}@urv.cat

Abstract

Enabling private relationships in social networks is an important issue recently raised in the literature. We describe in this paper a new protocol which offers private relationships allowing resource access through indirect relationships without requiring a mediating trusted third party (although an optimistic trusted third party is used which only acts *in case of conflict*). Thanks to homomorphic encryption, our scheme prevents the resource owner from learning the relationships and trust levels between the users who collaborate in the resource access. In this way, the number of users who might refuse collaboration due to privacy concerns is minimized. This results in increased resource availability, as the chances that certain nodes become isolated at a given period of time are reduced. Empirical evidence is provided about the proposed protocol being scalable and deployable in practical social networks.

Key words: Network security and privacy, Network services and applications, Social networks
1991 MSC: 91D30, 94A60

[1] The authors are with the UNESCO Chair in Data Privacy, but they are solely responsible for the views expressed in this paper, which do not necessarily reflect the position of UNESCO nor commit that organization.

[2] This work was partly supported by the Spanish Ministry of Science and Education through projects TSI2007-65406-C03-01 "E-AEGIS" and CONSOLIDER INGENIO 2010 CSD2007-00004 "ARES", and by the Government of Catalonia under grant 2005 SGR 00446.

1 Introduction

Social networks have become an important web service [1] with a broad range of applications: collaborative work, collaborative service rating, resource sharing, searching new friends, etc. They have become an object of study both in computer and social sciences, with even dedicated journals and conferences. They can be defined as a community of web users where each network user can publish and share information and services (personal data, blogs and, in general, resources). In some social networks, users can specify how much they trust other users, by assigning them a trust level [2,3]. It is also possible to establish several types of relationships among users (for example, "colleague of", "friend of ", etc.). The trust level and the type of relationship are used to decide whether access is granted to resources and services being offered.

As pointed out in [4,5], the availability of information on relationships (trust level, relationship type) has increased with the advent of the Semantic Web and raises privacy concerns: knowing who is trusted by a user and to what extent discloses a lot about that user's thoughts and feelings. See [6] for an analysis of related abuses.

These privacy issues have motivated some social networks [7,8] to enforce simple protection mechanisms, according to which users can decide whether their resources and relationships should be public or restricted to themselves and those users with whom they have a direct relationship. Unfortunately, such straightforward mechanisms result in too restrictive policies.

In [9], a more flexible access control scheme is described, whereby a *requestor* can be authorized to access a resource even if he has no direct relationship with the *resource owner*, but he is within a specified depth in the relationship graph. *Access rules* are used, which specify the set of *access conditions* under which a certain resource can be accessed. Access conditions are a function of the relationship type, depth and trust level. Relationship certificates based on symmetric-key cryptography are used by a requestor to prove that he satisfies some specific access conditions. To access resources held by a node with whom the requestor has no direct relationship, the requestor retrieves from a central node the chain of relationship certificates along the path from the resource owner to himself. Clearly, the central node is a trusted third party, as it knows the relationships of all nodes in the network.

In [10] a mechanism to protect personal information in social networks is described where nodes in the network are anonymous and cannot be linked to specific users; in contrast, the data and the relationships are public, which might facilitate user re-identification.

An innovative privacy-preserving approach is described in [4] which leans on

the access model in [9] and focuses on relationship protection: a user can keep private that he has a relationship of a given type and trust level with another user. Relationship certificates are encrypted and are treated like a resource in their own right: access to a certificate is granted using a *distribution rule* for that certificate, where the *distribution conditions* to be satisfied by users wishing to access the certificate are specified. If a user satisfies the distribution rule for a certificate, he receives the corresponding symmetric *certificate key* allowing him to decrypt the certificate. In [4] a rather complex scheme is proposed to manage and distribute certificate keys. Encrypted certificates are stored at a central node; due to encryption, the central node does not have access to the cleartext certificates, so it does not need to be trusted in this respect. However, the central node needs to be trusted in the following aspects: i) trust level computation when several relationship certificates are chained (indirect relationship between a resource requestor and a resource owner); ii) certificate revocation enforcement when a relationship ceases to exist (the central node must maintain a certificate revocation list and inform the other nodes about new revocations).

In [11] a protocol is proposed which overcomes the shortcomings detected in [4]. Specifically, the author presents a public-key protocol which achieves relationship protection without the presence of a central node working as Trusted Third Party (TTP). In addition to that, this protocol avoids revealing the content of relationships to the resource requestor and substantially simplifies relationship revocation. Nevertheless, this scheme has some shortcomings that we next summarize:

- For each resource access, a user tries to get the backing of the nodes with whom he is related. If a related node is temporarily unreachable or refuses to collaborate, it is hoped that other nodes related to the requestor will be available to act as intermediate nodes. However, a user with a small number of relations is likely to stay isolated at certain periods of time (*e.g.* early in the morning). This issue is an open problem which must be addressed.
- The protocol in [11] prevents the resource requestor from seeing any of the relationship certificates that will be used by the resource owner to decide whether the requestor is granted access. However, the resource owner learns the relationships, and their trust level, between the users who collaborate in the resource access. This represents a major privacy toll which would justify that some intermediate nodes might refuse collaboration. This fact also has implications for the previous point explained above: nodes which refuse to collaborate add to nodes which are unreachable and both categories disrupt in the same way the normal network operation.

1.1 Multiplicative privacy homomorphisms

Privacy homomorphisms (PHs) are encryption transformations mapping a set of operations on cleartext to another set of operations on ciphertext. Basically, PHs are encryption functions $E : CT \rightarrow CT'$ allowing a set F' of operations on a ciphertext domain CT' to be carried out without knowledge of the decryption function D. Knowledge of D allows the result of the corresponding set F of operations on a cleartext domain CT to be retrieved. A PH is called *multiplicative* when its set F of cleartext operations contains multiplication. A PH is called *probabilistic* if the encryption algorithm E involves some random mechanisms to choose the ciphertext corresponding to a given cleartext from a set of possible ciphertexts.

Privacy homomorphisms that will be used in our proposal below must be multiplicative, probabilistic and public-key. ElGamal [12] is a probabilistic public-key cryptosystem of integers in the multiplicative group $\mathbb{Z}_p^*$, where p is a large prime. This cryptosystem has a multiplicative homomorphic property which fulfills all these requirements.

1.2 Contribution and plan of this paper

Enabling private relationships in social networks is an important issue raised in [4] and extended in [11]. We describe in this paper a new protocol which offers the same features of [4] and [11] while providing a solution which addresses the drawbacks left open in [11]. However, such shortcomings are not solved without cost: we assume the existence of an optimistic TTP which only acts in case of conflict between the users of the social network. Such authority is not needed during the normal network execution. Therefore, we argue that this solution performs better than a (non-optimistic) TTP mediating all access requests.

Our scheme prevents the resource owner from learning the relationships and the trust levels between the users who collaborate in the resource access. In this way, the privacy threat detected in [11] is solved and the number of users who might refuse collaboration due to privacy concerns is minimized. As a result, the chances for certain nodes to become isolated at certain periods of time are reduced.

Section 2 describes the new protocol. In Section 3 its security is analyzed. Section 4 studies the performance of the proposed protocol using simulations. Finally, Section 5 is a conclusion.

2 Homomorphic access control protocols

We follow the framework from [4] modified according to [11], that is, we consider that the node owning a resource rid (hereafter, the *resource owner*) establishes an access rule $AR = (rid, AC)$ where AC is the set of access conditions to be simultaneously satisfied to access rid. Several alternative access rules can be defined for a resource. An access condition is a tuple $ac = (v, rt, t_{min})$ where v is the resource owner. Such node must have a direct or indirect relationship with the node requesting resource rid (hereafter, the *requestor*), and rt, t_{min} are, respectively, the type and the minimum trust level that the relationship should have. The trust level t is a rational value such that $0 \leq t \leq 1$. We use a privacy homomorphism to encrypt the trust values contributed by the nodes in the social network. However, multiplicative homomorphisms are only available for integers in the current literature. According to that, we propose to encode rational trust values as integer fractions; the details of the coding are given in Section 2.1.

Even though we use access rules and access conditions in a way similar to proposals [4,11], note that we differ from such schemes in that we do not use the maximum depth of the relationship as a requirement in access conditions. We have eliminated it because knowledge of the depth might be used by the resource owner to infer the trust level of the relationships between the users who collaborate in the resource access. We argue that this does not represent any security loss for the scheme since the minimum trust level and the type of the relationship are conditions that should be enough to decide whether a certain user can get access to a certain resource.

Each user U_i in the network owns two key pairs represented by (SK_i, PK_i) and (SSK_i, PSK_i). The former key pair corresponds to a public-key probabilistic multiplicative privacy homomorphism and it is used to encrypt/decrypt. The latter key pair is used to sign/verify. SK_i and SSK_i are the private keys. The corresponding public keys, PK_i and PSK_i, are assumed to be known and accepted by all users who have some interest in getting in touch with U_i. In the rest of this paper, one-to-one communications are assumed to be confidential and authenticated (by properly using encryption and message authentication codes).

We agree with [11] in that access should be enforced based on the relationship path between requestor and resource owner that yields the maximum trust level. This differs from the ideas presented in [9,4] where the trust level is computed taking into account all paths between requestor and resource owner, which might lead to overprotection: a requestor with a highly trusted direct relationship to the owner might be denied access just because there is also a requestor-owner indirect relationship with low trust through a third user.

We next explain how to encrypt rational numbers used as trust levels by means of a homomorphism for integer values. Then, a simple version of our access control enforcement protocol is described in Section 2.2 which will help the reader to understand the new scheme. After that, the privacy problems that arise when using such a simple protocol will be discussed. Then, an enhanced solution will be described. Last but not least, it will be explained how the resource owner finally transmits his resource to the requestor (assuming the resource owner accepts the requestor).

2.1 Homomorphic encryption of rational values

As explained previously, users cannot encrypt rational numbers directly. According to that, we propose that users send fractions which will represent the real number linked to a certain trust value. As an example, a certain user U_i who wants to contribute with a trust value of $0.\hat{3}$ will send fraction $1/3$ instead of the rational number. Note that the numerator and the denominator of the fraction are integers which can be encrypted in two different ciphertexts. Two different users U_1 and U_2 can multiply their own trust values (represented as fractions α_{U_1}/β_{U_1} and α_{U_2}/β_{U_2} respectively) in the ciphertext domain by performing the following operation:

$$\frac{E(\alpha_{U_1}) \otimes E(\alpha_{U_2})}{E(\beta_{U_1}) \otimes E(\beta_{U_2})} = \frac{E(\alpha_{U_1} \cdot \alpha_{U_2})}{E(\beta_{U_1} \cdot \beta_{U_2})}$$

where $E()$ denotes the encryption of a certain value following the privacy homomorphism in use and $\otimes$ denotes the ciphertext operation of such a cryptosystem corresponding to cleartext multiplication. At the end of the protocol, a user who is able to decrypt both resulting ciphertexts ($E(\alpha_{U_1} \cdot \alpha_{U_2})$ and $E(\beta_{U_1} \cdot \beta_{U_2})$) can divide the two recovered integer values to obtain the trust value as a rational number.

Nevertheless, this proposal introduces a privacy vulnerability: the user who is able to decrypt the ciphertexts may gain some clues from the resulting numerator and denominator on which fractions have been used to compute the final trust value. Since such fractions correspond to the trust values contributed by the users of the network, we argue that such information disclosure should be prevented. To address this situation we propose that, prior to encryption, each user U_i generates a random value ω_{U_i} that will be used to hide into a jumble of factors the numerator α_{U_i} and the denominator β_{U_i} that represent U_i's trust. Such hiding process is performed by multiplying numerator and denominator by ω_{U_i}:

$$\frac{E(\alpha_{U_i} \cdot \omega_{U_i})}{E(\beta_{U_i} \cdot \omega_{U_i})}$$

where ω_{U_i} is generated by multiplying a random sample (without replacement) of the prime numbers between 1 and a parameter n. Each prime is selected for the sample with probability γ. Each selected prime is raised to the power of an integer randomly selected between 1 and x (this integer is different for each prime).

Our scheme relies on multiplicative privacy homomorphisms to preserve the privacy of the users. Assuming that the privacy homomorphism in use is defined in the group $\mathbb{Z}_p^*$, the multiplication of all encrypted values must yield a result below p; otherwise information loss will occur. For the ciphertext containing the multiplication of all numerators this means

$$\prod_{i=1}^{s}(\alpha_{U_i} \cdot \omega_{U_i}) < p \tag{1}$$

where s is the total number of users whose trust is multiplied. An inequality analogous to Expression (1) can be written for the ciphertext containing the multiplication of all denominators. We next discuss which are the proper values for the parameters involved in those inequalities.

2.1.1 Parameter selection

As stated above, our scheme requires that the multiplication of encrypted numerators, resp. denominators, yields a result below p. The numerator α_{U_i} and the denominator β_{U_i} are the two elements of the fraction which represents U_i's trust value. Both elements are integers in the range $[0, \ldots, k]$. Value k is selected depending on the accuracy desired for the trust levels. As an example, with $k = 100$ we will guarantee an accuracy of two decimals: 0.93 can be represented by 93/100 or by 15/16 (numerators and denominators must be equal to or less than k). In what follows, $k = 100$ is taken.

Expression (1) depends on parameter s, which represents how many trust values will be multiplied together. In Section 4 we argue that usually $s \leq 6$. Nevertheless, we will fix it to $s = 7$ to leave enough room for situations with more trust levels to be multiplied.

In terms of length, the worst case for ω occurs when all prime numbers in $[1, \ldots, n]$ are selected, and all of them are raised to the power of x (the maximum value). In order to compute the maximum length, we assume that all n numbers are selected (prime or not). According to that, $(n!)^x$ is an upper

Table 1
For $k = 100$ and $s = 7$, upper bound given by Expression (2) for different values of n and x

n	$s \cdot (\log_2(k) + x \cdot \log_2(n!))$				
	$x = 3$	$x = 5$	$x = 10$	$x = 15$	$x = 20$
3	100.79	136.98	227.45	317.93	408.40
5	191.55	288.25	529.99	771.73	1013.47
7	304.79	476.98	907.45	1337.92	1768.40
11	576.77	930.27	1460.53	2697.81	3581.58
13	729.76	1185.26	2324.02	3462.78	4601.53
16	975.76	1595.26	3144.02	4692.77	6241.53
17	1061.60	1738.32	3430.14	5121.96	6813.77

bound on ω. It means that $\prod_{i=1}^{s}(\alpha_{U_i} \cdot \omega_{U_i})$ will be at most $(k \cdot (n!)^x)^s$. The length of this expression is:

$$|(k \cdot (n!)^x)^s| = s \cdot \log_2(k \cdot (n!)^x) = s \cdot (\log_2(k) + x \cdot \log_2(n!)) \tag{2}$$

If we set the length of p to 1024 bits, Expression (2) must stay below 1024 bits. Given $k = 100$ and $s = 7$, Table 1 shows values of n and x such that this requirement holds. It can be observed that for $x = 10$ (which is high enough), value n can be taken up to 7. In addition to that, if we set $n = 11$ (which is high enough too), x can be increased up to 5. These results prove that the proposed method for real number encoding is feasible and works properly.

Note that the considered upper bound on ω is a loose one: all numbers (prime or not) in $1, \ldots, n$ are selected. An average length for ω considering only the prime numbers can be computed as

$$\sum_{\forall \text{ prime } i \in (1,\ldots,n)} \gamma \cdot x/2 \cdot \log_2(i)$$

2.2 Simple homomorphic protocol

We present in this subsection a first protocol using homomorphic encryption to guarantee relationship privacy.
Protocol 1 (Simple homomorphic protocol)**.**

(1) *The resource owner B advertises to the network a certain resource rid he wants to share. Such an advertisement is signed by B, and contains all the access rules $AR_1, \ldots, AR_r$ defined for rid. That is, $\{AR_1, \ldots, AR_r\}_{SSK_B}$.*

(2) *The requestor A is interested in rid. In order to gain access to that resource, A sees to it that the resource owner receives one or several relationship certificates proving that the requestor satisfies all access conditions corresponding to at least one of the access rules. Several cases can be distinguished depending on the relationship depth between the requestor and the resource owner:*

   (a) Depth 1. *A and B have a direct relationship, that is, A is related to B through a relationship of type rt and trust level t_{AB} represented by a tuple (rt, t_{AB}) and B is related to A through a relationship (rt, t_{BA}). Note that the relevant trust level here is t_{BA} (how much B trusts A) which is assumed to be* unknown *to A. In this case A directly asks B whether he is granted access to the resource on the basis of (rt, t_{BA}). If B evaluates that rt and t_{BA} satisfy the set of access conditions targeted by A, then A is granted access. Otherwise, A is required to resort to other direct relationships or indirect relationships.*

   (b) Depth 2. *If A and B have no direct relationships (or these are not enough to buy access to A) then A asks to all users with whom A is directly related whether they have direct relationships of the relevant type rt with B. Assume C is directly related to both A and B with relationship type rt. Then C sends to B a pair of messages $PK_B(rt), PK_B(t_{CA})$ encrypted under the public key of the resource owner B.*

   The trust values are rational values homomorphically encrypted as fractions as detailed in Section 2.1. According to that, $PK_B(t_{CA})$ corresponds to:

$$PK_B(t_{CA}) \Leftrightarrow \frac{PK_B(\alpha_{CA} \cdot \omega_{CA})}{PK_B(\beta_{CA} \cdot \omega_{CA})}$$

   However, for the sake of readability, we prefer to write $PK_B(t_{CA})$ instead of the above fraction. Note that the encryption of the relationship type rt is simpler, because it is not a rational number.

   After C has sent to B the pair $PK_B(rt), PK_B(t_{CA})$, C tells A that a message was sent to B, but does not reveal *its content. At this point B evaluates whether a relationship of type rt and trust level $t_{CA} \cdot t_{BC}$ is enough to grant access of A to rid.*

   (c) Depth 3. *If access at depth less than or equal to 2 cannot be obtained then A requests to users C directly related to him to attempt access with depth 2 on A's behalf: each C directly related to A contacts his other directly related users D about possible direct relationships between D and B (similarly to what A did in Step 2b). If a D with direct relationships to C and B exists, D must multiply his own trust value related to C times the current trust value which comes from C.*

To do that, D computes:

$$PK_B(t_{CA} \cdot t_{DC}) = PK_B(t_{CA}) \otimes PK_B(t_{DC})$$

where $\otimes$ denotes the ciphertext operation of the privacy homomorphism corresponding to cleartext multiplication. According to the homomorphic fraction coding described in Section 2.1, $PK_B(t_{CA}) \otimes PK_B(t_{DC})$ corresponds to:

$$PK_B(t_{CA}) \otimes PK_B(t_{DC}) \Leftrightarrow \frac{PK_B(\alpha_{CA} \cdot \omega_{CA}) \otimes PK_B(\alpha_{DC} \cdot \omega_{DC})}{PK_B(\beta_{CA} \cdot \omega_{CA}) \otimes PK_B(\beta_{DC} \cdot \omega_{DC})}$$

The former expression implies the following correspondence:

$$PK_B(t_{CA} \cdot t_{DC}) \Leftrightarrow \frac{PK_B(\alpha_{CA} \cdot \omega_{CA} \cdot \alpha_{DC} \cdot \omega_{DC})}{PK_B(\beta_{CA} \cdot \omega_{CA} \cdot \beta_{DC} \cdot \omega_{DC})}$$

Finally, D sends a message containing $(PK_B(rt), PK_B(t_{CA} \cdot t_{DC}))$ to B. Upon receiving this message, B evaluates whether a relationship of type rt and trust level $t_{CA} \cdot t_{DC} \cdot t_{BD}$ is enough to grant access of A to rid. Note that B receives the product $t_{CA} \cdot t_{DC}$, but he cannot discover the individual trust levels which have been multiplied.

Figure 1 represents a path between the requestor A and the resource owner B through the social network. The picture shows the encrypted trust value as computed on its way from A towards B. The ciphertext containing the relationship type (rt) has been omitted.

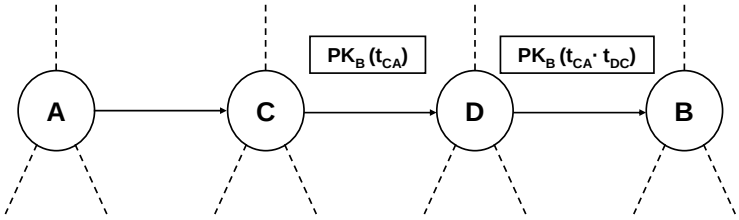

Fig. 1. Resource request in the simple homomorphic protocol

(d) Successive depths. *In case of failure at depth 3, successive depths are tried in a similar way.*

Remark. When the resource owner advertises the access rules for a resource, the access conditions in those rules leak the relationships the owner is involved in (*e.g.* if the owner accepts rt = 'Colleague at company X' this means that he works at Company X). In [4] the relationship type is kept confidential through a rather complex symmetric encryption scheme. A first problem of this scheme arises when the same relationship type is encrypted using two different keys by two different user communities and these merge at a later stage; another, perhaps more serious problem is how to revoke the key used

to encrypt a given relationship type. An alternative and simpler strategy is to "camouflage" the real relationship types among a large number of bogus relationships; then access conditions are published some of which use real relationship types and most of which use bogus relationship types. A bogus relationship type rt' is one that has never been established by the owner with anyone, so that no one can request access based on rt'. The advantage is that a snooper cannot tell bogus relationships from real ones, so that he does not know which relationships the owner is actually involved in.

2.3 Anonymous homomorphic protocol

Protocol 1 above is unsafe if nodes are not anonymous and there is a relationship of depth 2 between requestor and owner. We next explain why. Let A be a requestor, C be an intermediate node and B be a resource owner. During the protocol B gets $PK_B(t_{CA})$ and decrypts it to obtain t_{CA}, that is, the trust level assigned by C to A. This represents an unavoidable privacy problem in Protocol 1, as B needs to compute $t_{CA} \cdot t_{BC}$ in order to evaluate whether A is trusted enough to access a resource. Of course, one might argue that B does not know how many nodes there are between C and A (there might be several, and t_{CA} might be the product of the trust levels between C and A); however, this seems a rather weak protection, because nodes are not anonymous. Worse yet, the above weakness can easily spread. Once t_{CA} is learnt by B, for any path of length 3 D-A-C-B, then B will learn t_{AD}. And so on.

To avoid that, we propose to enforce anonymity for nodes which are not directly related. For example, in a path of length 3 A-C-D-B, this kind of anonymity means that B only knows D and he has no knowledge about which nodes or how many nodes are behind D; also, D only knows C, but not A. Furthermore, also for the sake of anonymity, C should not know that A is the requestor; in order to make himself undistinguishable from an intermediate node, A sends to C an encrypted trust value, which in A's case must be an encryption of the initial neutral trust 1, that is $PK_B(1)$ (note that, since we are assuming the use of probabilistic homomorphic encryption, C cannot discover that $PK_B(1)$ is an encryption of the maximum trust 1).

As mentioned above, node anonymity was previously proposed in [10]. However, node re-identification in that scheme was still possible because relationship types and trust levels were public. In our scheme, we combine node anonymity with encryption of trust levels, in order to provide real privacy for the users of the social network.

A problem which arises is that dishonest users may take advantage of anonymity to disrupt the system without being punished. We propose a *liability*

mechanism to thwart such disruption.

2.3.1 Liability mechanism

In our scheme, anonymity exists in indirect relationships (with depth greater than one), but direct relationships are non-anonymous: each node knows his directly related nodes. Furthermore, we require that every pair of directly connected nodes A and C should not only know but authenticate each other before engaging in any protocol transaction between them. As a result, a dishonest user E cannot impersonate a certain node of the social network (for example node A) and she cannot repudiate being user E.

According to that, an intermediate node C directly related to (and thus authenticated by) a resource owner B can be held liable by B for any harm that results from the encrypted trust level that C forwards to B. In turn, C can extend this liability to his direct relationships along the requestor-owner path. This liability transmission is the same used in chained subcontracting in daily life (*e.g.* the first subcontractor is liable in front of the main contractor, the second subcontractor is liable in front of the first subcontractor and so on).

Now, let us imagine that a resource owner B has followed the protocol and he has given an anonymous node access to a certain resource (*e.g.* a movie). Later on B discovers that this resource is being unlawfully re-distributed over the Internet. Following our liability mechanism, B will point C as guilty since C is the only user in the requestor-owner path known by B. Then B will take proper countermeasures against C. In turn, C (if he is not the dishonest user) will do the same with the direct node he knows and so on up to the requestor.

This mechanism has a major problem: at the end of the process, all users in the path pay the consequences of the misbehavior by a single node. Therefore, this situation can discourage users from collaborating in resource access. To solve that, we add to our liability mechanism the use of certificates as a proof of the direct relationship between two users in the access to a certain resource. For example, in a path of length 3 A-C-D-B, user A will give a certificate $Cert_{AC}$ to C as a proof of their direct relationship in a certain resource access. In turn, C gives a certificate $Cert_{CD}$ to D and so on. A certificate for a relationship where node N_1 forwards the encrypted trust level to node N_2 is constructed as follows:

$$Cert_{N_1N_2} \leftarrow \{id||N_1||N_2||K_{N_1}(t_{N_1N_0})||time\_stamp\}_{SSK_{N_1}} \tag{3}$$

In Expression (3), id is the identifier assigned to a certain resource access (which is linked to a certain item); N_1 is the identity of the node who constructs and sends the certificate; N_2 is the receiver; N_0 is the node preceding N_1 and

$K_{N_1}(t_{N_1N_0})$ is the trust value that N_1 assigns to N_0 (encrypted under a secret key K_{N_1} only known by N_1). Note that when N_1 is the requestor, N_0 does not exist. In this case $t_{N_1N_0}$ is taken to be 1, the neutral value for multiplication. The certificate is signed by N_1 and contains a *time_stamp* which reflects when it was generated. The public key PSK_{N_1} for verification is known and accepted as valid by all nodes in the network.

A user U who receives a certificate must store it in a safe place. Later on, if someone in the relationship path misbehaves, U can use the certificate to prove his innocence. We next explain how these certificates are used to protect the system against a misbehaving node (whether it is a requestor or an intermediate node).

Figure 2 represents the same situation shown in Figure 1 but following our proposed anonymous homomorphic protocol. Figure 2 shows a resource request from the point of view of the resource owner (node B). In this situation, B only knows that the computed trust value contains a trust from D ($t_{D?}$) but he does not know who is behind D. Actually, B does not know how many nodes are behind D neither who is the requestor.

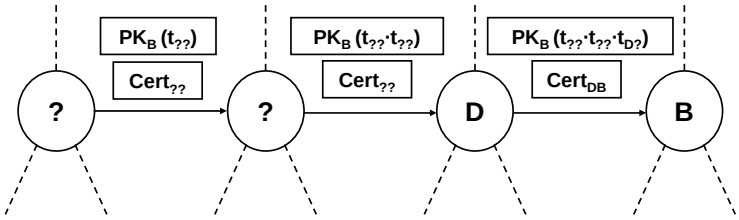

Fig. 2. Resource request in the anonymous homomorphic protocol (from the point of view of the resource owner)

Regarding the computational cost, our anonymous homomorphic protocol using the proposed liability mechanism requires intermediate nodes involved in a protocol execution to perform the following operations:

(1) Use the privacy homomorphism to encrypt a trust value t.
(2) Compute the ciphertext operation of the privacy homomorphism on the received encrypted trust value t' and the own encrypted trust value t.
(3) Generate the certificate $Cert_{N_1N_2}$ required by the liability mechanism.

A resource requestor only executes operations (1) and (3). A resource owner only decrypts the received ciphertexts containing the final trust value and the relationship type. Later, the resource owner checks the set of access conditions in plain text. Note that all these operations are feasible in full-fledged computers (*e.g.* desktop computers, laptops, powerful mobile devices, etc.). The social network that we envisage runs on this kind of devices so we argue that our protocol will perform properly in terms of computation when deployed in

a real environment.

2.3.2 Protecting the system by partial revocable anonymity

Nodes in the relationship path can misbehave in two different ways:

- A *requestor* who has gained access to a certain resource can later unlawfully re-distribute it over the Internet.
- An *intermediate node* can contribute false trust levels to the relationship path.

If the requestor misbehaves, an intermediate node U can use the certificate he owns to prove to the resource owner that he is only an intermediate node who has forwarded the encrypted trust level. In this way, node anonymity vanishes, because each intermediate node publishes his direct relationship in the path until the requestor is reached, who will be properly punished for his misbehavior. This procedure has a major problem: a dishonest resource owner can falsely pretend that a certain requestor has misbehaved in order to gain knowledge of all the relationships in the owner-requestor path.

We deal with this situation by proposing partially rather than totally revocable anonymity. According to that, we assume the existence of an *optimistic trusted third party* who is able to access the certificates to judge whether someone is misbehaving. In this way, the resource owner does not get any information on the requestor-owner path. An optimistic TTP is a trusted authority who only acts in case of conflict between the users of the social network. It is not needed during the normal network operation, which is much more efficient than requiring a (non-optimistic) trusted authority to mediate all transactions.

We now turn to the second type of misbehavior, in which an intermediate node contributes false trust levels to the relationship path. Even though trust values are kept secret, a certain user A having a direct relationship with another user B can make a good guess about the value of t_{BA}. There are two cases:

- If A relies on B's help to access resources and B misbehaves by contributing a fake t_{BA} *lower* than the real one, A will end up suffering a DoS attack. We believe that this situation should be avoided, or at least limited to the extent possible. Precisely to that end, Expression (3) includes the trust value $t_{N_1N_0}$ contributed by the node N_1 generating $Cert_{N_1N_2}$. Since $t_{N_1N_0}$ is encrypted under K_{N_1}, nobody but N_1 can recover $t_{N_1N_0}$. However, if a user thinks that he is facing a DoS attack from another user, he can report this situation to the optimistic TTP who requests the secret key used by each intermediate node; then, the optimistic TTP checks the trust values contributed by each intermediate node and takes action against those behaving dishonestly. Note that no one but this authority will know the trust value associated to each

direct relationship.

- If an intermediate node contributes a fake trust level *higher* than the original one, this can result in access being granted to a non-deserving requestor. Beyond illegal access, other problems may arise if the requestor unlawfully re-distributes the accessed resource or abuses it in other ways; see Subsection 2.3.1 above for a description of the defenses against a misbehaving requestor.

2.4 Accessing the resource

Our scheme is based on preserving node anonymity as long as nodes behave honestly. For that reason, a resource owner B who accepts to send a certain resource *rid* to a requestor D cannot transmit *rid* directly to D because the identity of the requestor is unknown to B. Instead of that, the resource must follow in reverse order the same relationship path which was previously used to decide on the requestor's access.

If this store-and-forward process involves too much bandwidth consumption for intermediate nodes (*e.g.* if the resource is a movie or some kind of large item), an alternative is for the requestor to send together with his query an ftp address where the resource owner can upload the resource. Note that this option implicitly assumes the existence of a trusted-third party: by default, ftp downloading is not anonymous and in this case the ftp site must be trusted to preserve the anonymity of all requestors who use it; if some anonymizer is used between the requestor and the ftp site, then this anonymizer is playing the role of a TTP who should preserve the anonymity of the requestor. Whatever the case, since the TTP can be totally external and unrelated to the social network or its users, assuming that such an entity will keep secret the identity of the requestor seems plausible.

3 Security analysis

We next explain the assumed adversary model and the possible attacks the system has to be robust against.

3.1 Adversary model

Our attacker model assumes that the adversary is a node in the social network. Such an adversary can collude with others to attack the system. We consider

that the computational power of an attacker does not permit him to break current computationally secure cryptosystems.

3.1.1 Attacks considered

We divide the possible attacks in four categories. The first three categories refer to the role adopted by the attacker: *intermediate node*, *resource requestor* and *resource owner*. The last category refers to *collusions between nodes*. We next list the attacks considered within each category.

- *Intermediate node.*
  - Learn the trust levels of the previous nodes in the requestor-owner path.
  - Alter the received trust or contribute a fake one.
  - Refuse to collaborate.
- *Resource requestor.*
  - Increase the trust sent by other users in order to get access to a certain resource.
- *Resource owner.*
  - Learn the trust levels of the nodes in the requestor-owner path.
- *Collusion between nodes.*
  - Learn the trust level of an honest user surrounded by colluders.

3.2 Protocol behavior against the considered attacks

3.2.1 Learn the trust levels of the nodes in the requestor-owner path

This attack can be performed by an intermediate node or by the resource owner. The resource owner is able to decrypt the encrypted product of trust levels and get the final trust value. However, the owner cannot learn from that value which nodes have collaborated nor the individual trust level contributed by each of them. An intermediate node is less dangerous than the resource owner, because he does not know the secret key needed to decrypt the received product of trust levels.

3.2.2 Alter the received trust or contribute a fake one

As said above, an intermediate node cannot learn the computed trust value received from his "upstream" neighbor (the neighbor previous to him in the requestor-owner path) because it is encrypted. However, the intermediate node can replace the received trust value with any trust value he desires and such a behavior will go undetected.

As explained in Section 2.3.1, our proposal uses certificates which contain the trust values contributed by each node. In addition to that, we assume the existence of an optimistic TTP who is able to check such values and take action against dishonest nodes. We argue that both measures will discourage users from performing this attack.

3.2.3 Refuse collaboration

An intermediate node can decide not to collaborate in a resource access. There is no defense against this. However, in our scheme there is no price to be paid by the intermediate nodes who collaborate: the privacy toll in [11] (mentioned in the introduction) is solved by our scheme and, with the use of an ftp service, an intermediate node is not even required to spend any of his bandwidth to enable resource access.

Therefore, intermediate nodes do not have any objective reason to refuse collaboration. In fact, there is even an incentive for them to collaborate: a node who routinely refuses collaboration will be in a bad position when he later seeks collaboration as a resource requestor.

3.2.4 Unlawful trust increase

The resource requestor cannot unlawfully increase any trust level in the requestor-owner path because he does not see any of the relationship messages exchanged between intermediate nodes that will be used by the resource owner to decide whether the requestor is granted access.

3.3 Collusion between nodes

Collusion between nodes will be successful when adversary nodes are surrounding the victim and the resource owner is one of the colluding nodes. In this situation the colluders will learn the trust value which the victim has assigned to his adversary upstream neighbor. However, we argue that the surrounded victim can decide not to collaborate if he does not trust his neighbors (upstream or downstream) or the owner (the identity of the latter is public, since he is offering a resource).

4 Simulation

We have simulated our protocol in a realistic environment to observe its performance. The network simulator *ns-2* [13] was used for this purpose.

Central to our proposal is the availability of on-line users who have relationships between them and enable a certain requestor to get access to a certain resource. Therefore, the lack of active users in the network is a potential problem. After pondering this issue, we decided to check the impact of the shortage of on-line users in small networks. For large networks with lots of connected users, the problem is less likely. With this in mind, the proposed scheme was tested in four social networks consisting, respectively, of 100, 300, 500 and 1000 users.

In each social network, the connection topology between users and their associated relationships were fixed as follows before starting the simulation:

- Each user was connected to (*i.e.* held relationships with) a number of users ranging between 1 and 30. The precise figure was decided using the *power-law distribution*. As stated in [14], typical social networks are reasonably well approximated using this distribution. As a result, in our simulated social network 78.76% of the users held a number of connections ranging between 1 and 15. Also, 31.07% of the users held only 1 or 2 connections.
- Each node could establish three different types of relationship with other nodes. Up to one relationship of each type was allowed between each pair of nodes, so that there could be up to three relationships between two nodes. Whether there existed a relationship of a certain type between two nodes was uniformly randomly decided.
- The trust level assigned to each existing relationship was randomly and uniformly chosen in the range $[0.5, 1]$.

A simulation test consisted of running a request by a requestor to access a resource advertised by an owner. Both requestor and owner were randomly chosen. The simulation ended when:

- either the requestor was granted access to the resource, which happened if the resource owner could compute a trust value greater than or equal to 0.4;
- or a time-out occurred (this happened when either no path existed between requestor and owner or the existing paths yielded a computed trust less than 0.4).

For each social network, several fractions of simultaneous on-line users were considered: 10%, 30%, 50%, 70% and 90%. For each fraction, 50 tests were run and the average results over the 50 tests were computed. The results for the four social networks considered are shown in Tables 2, 3, 4 and 5. The

contents of those tables are commented below.

Table 2
Results for a social network of 100 users

#on-line nodes	average access prob.	average # interm. nodes	average #messages	average trust level
10	0.09	2.89	192.91	0.42
30	0.20	3.10	886.72	0.48
50	0.36	2.94	1221.30	0.59
70	0.50	3.44	1635.24	0.50
90	0.70	3.29	1825.60	0.46

Table 3
Results for a social network of 300 users

#on-line nodes	average access prob.	average # interm. nodes	average #messages	average trust level
30 (10%)	0.11	3.33	260.00	0.50
90 (30%)	0.24	3.92	1298.25	0.49
150 (50%)	0.46	4.09	2493.65	0.45
210 (70%)	0.56	4.07	3009.82	0.42
270 (90%)	0.73	4.03	3616.33	0.44

Table 4
Results for a social network of 500 users

#on-line nodes	average access prob.	average # interm. nodes	average #messages	average trust level
50 (10%)	0.14	4.00	683.75	0.46
150 (30%)	0.28	3.43	1353.14	0.62
250 (50%)	0.49	5.36	2163.32	0.41
350 (70%)	0.66	5.33	3593.15	0.43
450 (90%)	0.82	4.59	3771.22	0.41

4.1 Simulation results

For each social network and each fraction of on-line nodes (*# on-line nodes*), the following results are reported:

Table 5
Results for a social network of 1000 users

#on-line nodes	average access prob.	average # interm. nodes	average #messages	average trust level
100 (10%)	0.18	5.20	815.37	0.41
300 (30%)	0.32	4.92	2880.25	0.46
500 (50%)	0.52	5.67	3373.04	0.42
700 (70%)	0.71	5.47	4412.25	0.43
900 (90%)	0.89	4.71	4628.02	0.42

- *Average access probability*. This is the average probability that a certain access request is granted (*i.e.* that a computed trust greater than or equal to 0.4 is obtained).
- *Average number of intermediate nodes*. This is the average number of intermediate nodes in the path between the requestor and the resource owner.
- *Average number of messages*. This is the average total number of messages generated by the nodes in the social network until the launched resource request is granted or a time-out occurs.
- *Average trust level*. This is the average trust level for the accepted requests.

From the results in Tables 2, 3, 4 and 5, it can be observed that better performance is obtained when there is a higher fraction of on-line users: the average access probability is higher and the requestor-owner path is shorter. This is what one would expect.

It can be seen in those tables that when 30% or less users are on-line it is difficult to get access to a resource. When 50% or more users are on-line the system works reasonably well. Note that these results are obtained in the worst possible scenario for our protocol: relatively small networks where a big proportion of the users have no more than 2 connections which have been randomly established. Under these circumstances, it is pretty likely that several nodes find themselves isolated (without connections to other on-line nodes). In a more realistic scenario with several human user communities with common interests, it would be more likely for nodes in the same community to be on-line at the same time and hold more connections between them. This similarity of habits would render node isolation rarer, even if there is a low fraction of on-line nodes.

Also, it can be observed that, whatever the proportion of on-line nodes, the average number of intermediate nodes in a requestor-owner path is no more than six. This should not be surprising, because social networks are affected by the "six degrees of separation" phenomenon [15]: long ago, S. Milgram experimentally showed that any two people in the United States are connected through

about six intermediate acquaintances, implying that we live in a rather small world. According to the "six degrees of separation" concept, for huge social networks the separation between any two users will still be a maximum of six intermediate nodes. This is good news for our scheme in the sense that the number of generated messages until a resource request is accepted does not grow linearly with the number of on-line nodes. In fact, we can observe in Table 5 that the total number of messages sent is quite similar when there are 300 and 900 on-line nodes. These results show that our proposal is scalable enough to work properly when deployed in real social networks.

5 Conclusion

Protecting the type and trust level of relationships is an important privacy issue in social networks. However, this problem has not been deeply addressed in the literature.

We have described in this paper a new protocol which achieves protection of relationship privacy, with the advantage of being fault-tolerant and free of mediating TTPs (although an optimistic TTP is used in case of conflict). On the whole, our scheme offers the same features as [4] and [11] while addressing the functionality and privacy drawbacks of those previous protocols. Besides, the simulated performance of our proposal shows that it can be successfully deployed in real environments because: i) it is scalable; and ii) it provides reasonable resource availability.

References

[1] S. Staab, P. Domingos, P. Mika, J. Golbeck, L. Ding, T. W. Finin, A. Joshi, A. Nowak and R. R. Vallacher, "Social networks applied", *IEEE Intelligent Systems*, vol. 20, no. 1, 2005, pp. 80-93.

[2] R. Ashri, S. D. Ramchurn, J. Sabater, M. Luck and N. R. Jennings, "Trust evaluation through relationship analysis", *4th International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM, 2005, pp. 1005-1011.

[3] J. Sabater-Mir, "Towards the next generation of computational trust and reputation models", *Modeling Decisions for Artificial Intelligence-MDAI*, LNCS 3885, Springer-Verlag, 2006, pp. 19-21.

[4] B. Carminati, E. Ferrari and A. Perego, "Private relationships in social networks", *Private Data Management*, IEEE Press, 2007.

[5] A. J. Mikroyannidis, "Towards a social semantic web", *Computer*, vol. 40, no. 11, 200, pp. 113-115.

[6] S. B. Barnes, "A privacy paradox: social networking in the United States", *First Monday*, vol. 11, num. 9,2006.

[7] Facebook, `http://www.facebook.com`

[8] Vidently, `http://vidently.org`

[9] B. Carminati, E. Ferrari and A. Perego, "Rule-based access control for social networks", *OTM Workshops*, LNCS 4278, Springer-Verlag, 2006, pp. 1734-1744.

[10] D.-W. Wang, C.-J. Liau and T. Sheng Hsu, "Privacy protection in social network data disclosure based on granular computing", *IEEE International Conference on Fuzzy Systems*, IEEE Computer Society, 2006, pp. 997-1003.

[11] J. Domingo-Ferrer, "A public-key protocol for social networks with private relationships", *Modeling Decisions for Artificial Intelligence*, LNCS 4617, Springer-Verlag, 2007, pp. 373-379.

[12] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, 1985, pp. 469-472.

[13] *The Network Simulator*,
`http://nsnam.isi.edu/nsnam/index.php/Main_Page`

[14] D. Liben-Nowell, "An algorithmic approach to social networks", Ph.D. Thesis, MIT Computer Science and Artificial Intelligence Laboratory, 2005.

[15] S. Milgram, "The small world problem", *Psychology Today*, vol. 2, 1967, pp. 60-70.