2011 35th IEEE Annual Computer Software and Applications Conference

P^{3}D - Privacy-Preserving Path Discovery in Decentralized Online Social Networks

Mingqiang Xue [*1], Barbara Carminati [#2], Elena Ferrari [#3]

[*]*Department of Computer Science, National University of Singapore, Singapore*

[#]*Department of Computer Science and Communication, University of Insubria, Italy*

[1]xuemingq@nus.edu.sg, {[2]barbara.carminati, [3]elena.ferrari}@uninsubria.it

***Abstract*—One of the key service of social networks is *path discovery*, in that release of a resource or delivering of a service is usually constrained by the existence of a path with given characteristics in the social network graph. One fundamental issue is that path discovery should preserve relationship privacy. In this paper, we address this issue by proposing a *Privacy-Preserving Path Discovery* protocol, called P^3D. Relevant features of P^3D are that: (1) it computes only aggregate information on the discovered paths, whereas details on single relationships are not revealed to anyone; (2) it is designed for a decentralized social network. Moreover, P^3D is designed such to reduce the drawbacks that offline nodes may create to path discovery. In the paper, besides giving the details of the protocol, we provide an extensive performance study. We also present the security analysis of P^3D, showing its robustness against the main security threats.**

I. Introduction

In the past few years, social network (SN) platforms have gained great popularity among Internet users. Interesting applications, e.g., games, resource sharing, e-markets, location-based services can be built on top of them. Common to many of these applications is the need of discovering paths in the SN. For instance, in an e-commerce scenario, a user may be entitled to have a discount for some services, provided that he/she holds a relationship (e.g., "friendOf") with an already subscribed user. Another example is given by resource sharing, which is one of the most widespread activities in a social network, as very often resources may contain sensitive information, users may only want to share them with certain groups of trusted users. An intuitive way of denoting such users is by posing constraints on the relationships that the requestor should have with the resource owner. Constrain resources sharing in SNs is further motivated by the emerging trend of using social networking functionalities at the enterprise level as a means to facilitate knowledge sharing and information dissemination [15]. In general, an enterprise-oriented SN has size smaller than a general purpose SN, like Facebook, but has stronger requirements for resources confidentiality and user privacy. As such, it represents a good scenario for our proposal.

Indeed, one fundamental issue of path discovery the privacy of personal relationships from both other network users and the network manager itself. To cope with this issue, in this paper we propose an innovative *Privacy-Preserving Path Discovery* protocol, called P^3D. P^3D is able to protect the fundamental properties of a path, that is, the relationship type, the depth, and the trust level. Relevant features of P^3D are that it computes only aggregate information on indirect relationships (i.e. trust, depth and relationship type), whereas details on single relationships are not revealed to anyone. A further benefit of P^3D is that it has been designed for decentralized SNs, that is, SNs where resource management is delegated to each single network node, rather than to a centralized social network manager. This paradigm shift from client-server to a peer-to-peer infrastructure is today envisioned as one of the most promising approach for a next generation of social networks [4], [11], [18]. Decentralization can provide answers to issues that have raised controversy in the context of centralized SNs, such as the ownership of personal information and the protection of privacy, problems in cross-platform service provision and user lock-in. Moreover, decentralization promises higher performance, fault-tolerance and scalability in the presence of an expanding base of users and applications.

P^3D achieves its goals through a collaborative protocol, where selected nodes in the network participate to path discovery. Moreover, P^3D is designed to reduce the drawbacks that offline nodes may create to collaborative path discovery. This is achieved through the notion of *virtual edges* that makes it possible to privacy-preserving reuse the outcome of previous P^3D executions.

The privacy-preserving features of P^3D that can be summarized as follows: (1) P^3D hides aggregate information on discovered paths to any node involved in the collaborative process, with the exception of the owner; (2) information on relationships traversed during the execution of the protocol is hidden to everyone; (3) information on the constraints on the path to be discovered (in terms of depth, trust and relationship type) is hidden to everyone, with the exception of the owner; (4) the identity of the requestor is hidden to everyone; (5) the identity of intermediate nodes is hidden to everyone (clearly with the exception of direct contacts).

II. Related Work

In [2] relationship privacy is protected by defining a set of distribution rules, which basically state who can access a given relationship. These rules are then enforced through cryptographic techniques. However, the enforcement approach is still centralized. An alternative decentralized solution has been proposed in [1]. The solution is, like P^3D, collaborative,

0730-3157/11 $26.00 © 2011 IEEE
DOI 10.1109/COMPSAC.2011.15

	Depth	RelType	Trust	Requestor anonymity	Offline nodes	Network model
B. Carminati *et al.*. [2]	No privacy	No privacy	No privacy	No	No	Semi-centralized
B. Carminati *et al.* [1]	No privacy	No privacy	No privacy	No	No	Distributed
J. Domingo-Ferrer [5]	No privacy	No privacy	Yes, but limited	No	No	Distributed
J. Domingo-Ferrer *et al.* [6]	No privacy	No privacy	Yes	No	No	Distributed
K.B. Frikken and P. Srinivas [8]	Yes	No support	No support	No	Yes	Distributed
G. Mezzour *et al.* [16]	Yes	No support	No support	No	Yes	Distributed
P^3D	Yes	Yes	Yes	Yes	Yes	Distributed

TABLE I
COMPARISON WITH OTHER PROPOSALS

but the collaboration is driven by the specified distribution rules. This means that all the nodes taking part in the collaboration are aware of the characteristics of the path being discovered as well as of the involved nodes. Moreover, the protocol is not able to avoid that a malicious user sends the collaboration request, and the related path information, to a user which does not satisfy the distribution rules associated with the path. Rather, a mechanism is provided that makes the user requesting path discovery able to detect a posteriori these malicious behaviors. In [5] a fully decentralized solution to path discovery has been proposed, based on public key cryptography. However, it suffers from a major privacy risk: the owner learns the types and the trust values of the relationships between the nodes that participated in the path discovery process. To address this shortcoming, an improvement to the protocol has been proposed in [6], making use of homomorphic encryption to encrypt relationship trust values. By exploiting homomorphic encryption, the owner can only see the aggregated trust value but not the trust values of each relationship in the path. We use the same idea in our protocol. However, a major drawback of [6] that we avoid in our protocol is that a malicious node in the path can increase the overall path trust value by simply contributing with an incorrect trust value, that is, a trust value that is greater than 1 without being detected. Another major difference between P^3D and the above-mentioned collaborative protocols, is that previous work have assumed that the nodes are always online to collaborate in the path discovery process, an assumption which is unrealistic in the context of SNs. The only other proposals we are aware of dealing with offline nodes are [8], [16]. In [16], two nodes not connected by a direct relationship can compute the depth of all the paths between them without referring to other nodes, by exchanging information which is pre-computed at an earlier flooding phase (i.e., tokens). However, the protocol is only able to deal with discovery of paths with a specific length, whereas relationship type and trust are not considered. In addition, this protocol could be inefficient for real SNs, in that the token flooding phase is inefficient for large SN. To overcome this problem, in our protocol we employ flooding optimizations to greatly reduce the number of broadcast messages making the process more efficient. Moreover, [16] uses a secure multi-party computation solution to determine the paths between two nodes. If the set of tokens owned by both nodes are large, which is true when the nodes have high connectivity as in real SNs, the computation is inefficient. In [8], an alternative approach is proposed able to manage offline nodes. More precisely, the resources posted by the owner are encrypted by specific keys. A requestor node can only derive the key for decrypting a particular resource if and only if the requestor is within d hops from the owner, where d is the owner defined access control parameter. Again, the limitation of this approach is that the decision on whether to release a resource can only rely on the path depth not on trust values nor relationship types. Most important, from the view of privacy protection, the proposed scheme does not prevent that the whole SN graph can be easily recovered by a malicious party. Table I highlights the differences between our approach and other related work. This shows that our approach offers more privacy properties and flexibility than previous proposals.

III. BACKGROUND

An SN can be represented as a graph, where each node denotes a user in the network, whereas edges represent the existing relationships and their trust levels. Edge direction denotes which node specified the relationship and the node for which the relationship has been specified.

Definition 1: **Social network graph.** A social network graph is a tuple $(V_{SN}, E_{SN}, RT_{SN}, \phi_{SN})$, where RT_{SN} is the set of supported relationship types, V_{SN} and $E_{SN} \subseteq V_{SN} \times V_{SN} \times RT_{SN}$ are the nodes and edges of a directed labeled graph $(V_{SN}, E_{SN}, RT_{SN})$, whereas $\phi_{SN} : E_{SN} \rightarrow [0,1]$ is a function assigning to each edge $e \in E_{SN}$ a trust level, which is a rational number in the range [0,1].[1]

An edge $e = (v, v') \in E_{SN}$ expresses that node v has established a relationship with node v', denoted by $rel(v, v')$, where $rt_{rel(v,v')} \in RT_{SN}$ represents the relationship type associated with $rel(v, v')$. Moreover, we say that relationship $rel(v, v')$ is *direct* since v and v' are directly connected by edge e. Between two nodes v and v' there may also exist an *indirect* relationship of type rt, that is, a relationship consisting of a path connecting v and v', where all edges have the same direction and type rt. Hereafter, given a relationship $rel(v, v')$ we define as *depth* of $rel(v, v')$ (i.e., $d_{rel(v,v')}$) the number of edges in the path connecting v and v' in $rel(v, v')$. Besides of relationship types and depth, a relationship $rel(v, v')$ has a further property, i.e., its *trust* ($tv_{rel(v,v')}$). Since the trust level of a relationship is sensitive information, we assume that it is only known by the user establishing the relationship, not by the user connected through the relationship. The trust of direct relationship is given by the trust value associated with the

[1]For simplicity, we assume that trust level is specified as value in [0,1]. However, more general domain, like strings (e.g.,{"not trusted", "trusted", "very trusted"}), can be adopted and then normalized to [0,1] (e.g., {0,0.5,1}).

unique edge representing the relationship. Trust computation for indirect relationship has to take into account that between two nodes there may exist multiple paths denoting the same indirect relationship. Literature indeed offers several algorithms to compute the trust value of indirect relationships consisting of multiple paths. These algorithms mainly differ on how they select the paths to be considered for trust computation (see [10] for a survey). As the main goal of our protocol is to discover paths, without constraining the trust algorithm, in this paper, for simplicity, we assume that the trust of indirect relationships is computed as the product of the trust values of all the edges in a single path connecting the involved nodes. Such method has also been used in the D-FOAF system [13] [3]. However, it is relevant to note that the proposed path discovery protocol can potentially discover all paths connecting two nodes, therefore any trust algorithm could be applied.

We assume that release of a service/resource is governed by one or more *access rules*, specified according to the model in [2]. An access rule is a set of *access conditions*, specifying the requirements to be all satisfied for accessing the service/resource. Such requirements are expressed in terms of paths in the SN graph. An access condition is a tuple $(owner, rt, dmax, tmin)$, stating that between the resource owner/service provider and the requesting user there should exist a relationship of type rt, with maximum depth equal to $dmax$ and having a trust level greater than or equal to $tmin$. In the following, we use the term resource to denote either a service or a resource. An example of access condition regulating resource releasing in enterprise-oriented SNs is $(Bob, ColleagueOf, 3, 0.6)$, which states that the associated resource can be accessed only by nodes connected with Bob with a direct or indirect colleague relationship, with maximum depth 3 and 0.6 as minimum trust value. In the paper, for simplicity, we assume that resource releasing is governed by only one access condition AC, as the extension of P^3D to deal with more access conditions is trivial. Finally, we denote with rt_{AC}, $depth_{AC}$, $trust_{AC}$, the relationship type, depth and trust specified in the access condition AC.

IV. Privacy-Preserving Path Discovery

The basic idea of P^3D is that the owner initializes a set of messages and forwards them to the requestor prior to path discovery. Upon receiving the messages, each requestor updates and forwards the messages to start path discovery process. The collected data are stored into a set of privacy-preserving data structures (called *tokens*) Each intermediate node updates the received tokens to encode relationship information including relationship type, trust value and depth count, but it is unable to learn private information from the tokens. Eventually after receiving the updated tokens, the owner decodes the messages and derives information about the relationships with the requestor.

Let us see in more details how P^3D works. The process is initiated when a requestor node v_r requires a resource to another node v_o (the owner). Since an essential property of our proposal is to hide the requestor identity, we assume that the real IP address of v_r is hidden from v_o so that v_o cannot associate the IP address of the request packet to the identity of v_r. This property can be easily achieved by using an anonymous proxy (e.g., www.zend2.com, www.proxify.com, tproxy.guardster.com) to mediate between v_r and v_o. Alternatively, anonymity network such as Tor[2] can be also used. We would like to emphasize that since each v_r can randomly choose its own proxy server or anonymity network, therefore, this does not create a bottleneck for communication or scalability. To protect the confidentiality of messages exchanged between the owner and requestor, we assume that each message is encrypted by a shared key SK, previously agreed by them using a key generation protocol such as secure Diffie-Hellman key exchange [12]. Finally, each node v is provided with a pair of homomorphic public/private keys, denoted as $(v.pk, v.sk)$.

Once the owner receives a request, it retrieves the *access condition* AC regulating the resource release and initializes the tokens. Each path property is stored into a distinct token (i.e., *depth token*, *trust token*, and *relationship type token*, respectively) to which we collectively refer to as *tokenset*. Tokens are then forwarded to the requestor,[3] which, as a consequence, starts the collaborative process. The requestor forwards the tokens to those of its neighbors that have established a direct relationship with it. These nodes update the received tokens with information on depth, trust and relationship type of the traversed edge and forward the updated tokensets. The process is iterated until the owner receives back the tokens. By using the relationship type token, the owner is able to determine whether all the edges in the path have a relationship type equal to the one required by the access condition. In case the edge relationship types are different, the owner is not able to determine their real values. In contrast, by exploiting the depth and trust tokens the owner can determine the number of edges in the path, and the trust value of the indirect relationship, without knowing the trust value of any edge in the path.

In the following, we explain in more details how the tokenset is initialized, updated and used to compute the corresponding path properties. In doing this, we use the following notations. Given an instance of P^3D, we denote with $DPaths$ the set of discovered paths, where for any $p \in DPaths$, we denote with TN_p the sequence of nodes that P^3D has traversed to discover path p. We refer to this sequence as the *traversed nodes*, $v_j \in V_{SN}$, where $j = \{0, 1, \ldots, |TN_p|\}$. For any $p \in DPaths$ the following correspondences always hold: $v_0 = v_o$; $v_1 = v_r$ and $v_{|TN_p|} = v_o$.

A. Depth computation

The idea underlying the depth computation is very simple and makes use of only an hash function $H()$. The owner initializes the depth token with the hash value of a random number r_d. Then, starting from the requestor, each node traversed during P^3D updates the received depth token by simply hashing the hash value contained into it. Later in Section VII we assume a semi-honest adversary model implying that any

[2]http://www.torproject.org/.

[3]As the owner does not know the requestor identity, this forwarding is mediated by the anonymous server.

intermediate node follows the execution of the protocol, but it is curious in gaining additional information, as such we assume that each node performs the hash on the received depth token only once. When the owner receives the depth token back, it is able to calculate the number of edges in the path by iteratively computing the hash value of $H(r_d)$ until it is equal to the one stored into the received depth token. The number of iterations corresponds to the number of edges in the path. The depth token can therefore be formally defined as follows.

Definition 2: **Depth token**. Let $p \in DPaths$ be a path discovered during P^3D. For any, $v_j \in TN_p$ the depth token generated by v_j, denoted as $depth\_tk_j$, is defined as:

* if $j = 0$: $depth\_tk_0$=$H(r_d)$, where r_d is a random number generated by v_o;

* if $j = 1$: $depth\_tk_1$=$depth\_tk_0$;

* if $1 < j < |TN_p|$: $depth\_tk_j$=$H(depth\_tk_{j-1})$, where $depth\_tk_{j-1}$ is the depth token received from node $v_{j-1} \in TN_p$.

B. Trust computation

To support privacy-preserving trust computation, we exploit homomorphic encryption (as also done in [6], [16]). However, differently from previous work, we ensure that trust values aggregated during the process are correct w.r.t. the trust values domain, that is, they belong to the range [0,1]. This property is important since it avoids that possible malicious nodes can contribute to the trust computation with values greater than 1, to make the final product meeting the trust constraint in the access condition. Note that this bad behavior cannot be avoided just assuming a semi-honest adversary model, as a node is free to change the trust value of established relationship as it wishes. However, the adopted trust computation strategy avoid fake trust values, that is, values greater than 1. To obtain this, we propose the *non-increasing multiplication*, ensuring that the intermediate products are always non-increasing along the path. This operation is based on a *non-decreasing addition*, that is, an addition operation ensuring that only non-negative values are added to the result, so not to decrease it. In P^3D, we implement a non-decreasing addition by using an hash function as follows. Suppose that a node Y wishes to add a number l to a given value k, generated by X. To ensure that l is non-negative, X first creates a secret m, then it recursively computes an hash function $H()$ over m for k times obtaining $H_k(m)$. To add a number l to k, Y can recursively hash $H_k(m)$ for l times, obtaining $H_{k+l}(m)$. The final result $k+l$ can be obtained by X, by counting the number of hashing over m needed to get $H_{k+l}(m)$. This scheme allows Y to either hash or not to hash, in either cases it cannot decrease the counting, which satisfies the *non-decreasing addition* property.

P^3D *non-increasing multiplication* is implemented as follows. Suppose X generates a number k and a node Y wishes to multiply the value l to k. X first creates $\log_a k$ where $a \in (0,1)$, and sends it to Y. Y computes $\log_a l$ and adds it to $\log_a k$. This addition is a *non-decreasing addition* $+_a$, such that $k +_a l = \log_a k + \log_a l$. Y sends the value $k +_a l$ back to X, and X computes the value of $k \cdot l$, by raising a to the power of $\log_a k \cdot l$, as according to the logarithm properties, $k +_a l = \log_a k \cdot l$. Note that, since $k +_a l = \log_a k +_a \log_a l = \log_a k \cdot l$, then $\log_a k \cdot l > \log_a k$, so $k \cdot l < k$ due to the property of logarithm function with $a \in (0,1)$.

Therefore, the owner generates a random value r_t, and initializes the trust token with $H(r_t)$. Once received, the requestor simply forwards it as it is to the next node in the traversed path. When an intermediate node v_j receives the trust token: (1) it first retrieves the trust value t of the edge connecting it with the node from which it received the trust token, then (2) it computes $\log_a t$, and multiply the result by a scaling factor n_{sc} such to obtain an integer value $LogT$.[4] As a final step (3), v_j computes the hash function $H()$ over the hash value contained in the received trust token for $LogT$ times. This step performs the $+_a$ addition. The resulting value is then passed to the next nodes. Once the owner receives the final trust token, to retrieve the hidden product, it counts how many hash iterations have to be evaluated on r_t to reach the hash value contained in the token. This number represents the result of a non-decreasing addition (i.e., $k+_a l$), thus by raising a to the power of it, the owner obtains the product of the non-decreasing multiplication (i.e., $k \cdot l$), that is, the multiplication of trust values in the path. We can now formally define the trust token.

Definition 3: **Trust token.** Let $p \in DPaths$ be a path discovered during P^3D. Let a be the base used in P^3D. For any, $v_j \in TN_p$, the trust token generated by v_j, denoted as $trust\_tk_j$, is defined as:

* if $j = 0$: $trust\_tk_0$=$H(r_t)$, where r_t is a random number generated by v_o.

* if $j = 1$: $trust\_tk_1$=$trust\_tk_0$;

* if $1 < j < |TN_p|$: let $rel(v_j, v_{j-1})$ be the relationship according to which node $v_{j-1} \in TN_p$ sent to $v_j \in TN_p$ the trust token $trust\_tk_{j-1}$. Then,
$trust\_tk_j$=$H_{\lfloor n_{sc} \cdot log_a(tv_{rel(v_j, v_{j-1})}) \rceil}(trust\_tk_{j-1})$.

In certain cases, the trust value of a relationship can be 0 and the logarithm of the trust value is not defined. In our algorithm, such trust token is replaced with a random number of the same length as the trust token that encodes a non-zero trust value. Then with high probability, the random number is distinct from $H^k(r_d)$, for any $k = 0, 1, \ldots, k_{max}$ where k_{max}. After the owner has received the trust token and the number of hashing iterations is larger than k_{max} before finding the trust value, the owner conclude that the trust value of the path is 0.

C. Relationship type computation

P^3D aims to make the owner able to verify if all the edges in the discovered path have the type equal to the one required in the access condition, by at the same time protecting the relationship privacy. This implies to: (1) hide to all the nodes traversed during path discovery the relationship types

[4]We assume that the scaling factor as well as the base are publicly available to SN nodes. Note also that due the rounding operation the aggregate trust value could have some inaccuracy. However, by properly setting parameters a and n_{sc} this error can be limited as shown in [19].

of involved edges; (2) make the owner aware of the types of the relationships in the path only if these are all equal to the one required in the access condition, i.e., rt_{AC}. To achieve this, the owner initializes the relationship type token with the encryption of rt_{AC}. This encryption is computed by using the homomorphic public key of the owner. Due to the properties of homomorphic encryption, the intermediate nodes are able to participate in token computation without decrypting it, which ensures property (1). Moreover, the proposed scheme is defined such that the owner is able to decrypt it to a meaningful value only if all the edges in the path are of the same type as rt_{AC}. If this is not the case, the decryption results in a random number that does not reveal the relationship types in the path, which ensures property (2). In order to devise an encryption scheme satisfying property (2), we need to have a function, hereafter called $\sigma()$, satisfying the following properties:

$$\sigma(s_1, s_2) = \begin{cases} \pi & \text{if } s_1 = s_2 \\ r & \text{if } s_1 \neq s_2 \end{cases} \quad (1)$$

The $\sigma()$ function compares two numbers s_1 and s_2, and outputs π if they are equal and r, otherwise, where π is a particularly chosen integer parameter from a large prime cyclic group and r is an integer, randomly and uniformly picked up from the same group. According to this definition, the probability that $r = \pi$ is negligible. A possible implementation of $\sigma()$is the following:

$$\sigma = (\frac{s_1}{s_2})^{\epsilon} \cdot \pi \mod p \quad (2)$$

where p is a large prime number, and ϵ is a uniformly random exponent.In the following, we show how we can combine the σ function together with homomorphic encryption to obtain a scheme satisfying properties (1) and (2). To this purpose, we exploit ElGamal encryption [7].

Let $p = 2^k \cdot q+1$ be a prime for prime q and a non-trivial natural number k. We assume that the resource owner is equipped with a pair of ElGamal encryption key $(v_o.sk, v_o.pk)$, where $v_o.pk = g^{v_o.sk} \mod p$ for the generator g. According to the ElGamal encryption scheme, the encryption of a message m is a pair (m_1, m_2), where $m_1 = m \cdot (v_o.pk)^r$, $m_2 = g^r$, and r is a random number in $\{0, \ldots, q-1\}$. The decryption is computed as $m = \frac{m_1}{m_2^{v_o.sk}}$. Further, we assume that each relationship type in the system is encoded as a unique integer number. For simplicity, in the following, a relationship type rt_{AC} has to be intended as the corresponding integer number.

Let us now show how the relationship type token is computed during the discovery of a given path p. For the simplicity of our illustration, we set $\pi = 1$ for the following example. v_o initializes the relationship type token with $C_0 = (rt_{AC} \cdot v_o.pk^{r_0}, g^{r_0})$ and $E_0 = (\pi \cdot v_o.pk^{r'_0}, g^{r'_0})$, where r_0 and r_0 are random numbers, r_0 selected from $\{0, \ldots, q-1\}$. Then, it sends the token to the requestor which sets $C_1 = C_0$, $E_1 = E_0$, and forwards it to its direct neighbor, i.e., $v_2 \in TN_p$, where p is the traversed path. Upon receiving C_1, v_2 (1) computes $C_2 = (rt_{rel(v_2,v_1)} \cdot v_o.pk^{r_2}, g^{r_2})$ and $C_{2/1} = (\frac{rt_{rel(v_2,v_1)}}{rt_{AC}} \cdot v_o.pk^{r_1-r_2}, g^{r_1-r_2})$, where $rt_{rel(v_2,v_1)}$ represents the relationship type (i.e., its integer encoding) of the edge between v_2 and v_1; (2) v_2 generates a random exponent ϵ_2 and computes $E_2 = C_{2/1}^{\epsilon_2} = ((\frac{rt_{rel(v_2,v_1)}}{rt_{AC}})^{\epsilon_2} \cdot v_o.pk^{(r_2-r_1)\cdot\epsilon_2}, g^{(r_2-r_1)\cdot\epsilon_2})$. For simplicity, the randomization parameter used in the ElGamal encryption in E_j is always denoted as r. Thus, $E_2 = ((\frac{rt_{rel(v_2,v_1)}}{rt_{AC}})^{\epsilon_2} \cdot v_o.pk^r, g^r)$. Then, node v_2 forwards C_2 and E_2 to its direct neighbor (i.e., $v_3 \in TN_p$). Let us consider the generic step j, where $3 \leq j < |TN_p|$. Let $rel(v_j, v_{j-1})$ be the relationship according to which node v_{j-1} sent to v_j the relationship type token. v_j computes the following components: (1) $C_j = (rt_{rel(v_j,v_{j-1})} \cdot v_o.pk^{r_j}, g^{r_j})$, where $r_j \in \{0, \ldots q-1\}$ is a random number generated by v_j; (2) $E_j = C_{j/(j-1)}^{\epsilon_j} \cdot E_{j-1}$ $= ((\frac{rt_{rel(v_j,v_{j-1})}}{rt_{rel(v_{j-1},v_{j-2})}})^{\epsilon_j} \times (\frac{rt_{rel(v_{j-1},v_{j-2})}}{rt_{rel(v_{j-2},v_{j-3})}})^{\epsilon_{j-1}} \times \ldots \times (\frac{rt_{rel(v_2,v_1)}}{rt_{AC}})^{\epsilon_2} \cdot v_o.pk^r, g^r)$, where $C_{j/j-1} = \frac{C_j}{C_{j-1}}$, and ϵ_j is a random number generated by v_j. As final step, the owner computes $E_{|TN_p|}$, which is an encrypted token that aggregates relationship information of all the edges along the path. Since the owner owns the private key $v_o.sk$, it is able to decrypt $E_{|TN_p|}$ and compute the following σ value:

$$\sigma_{|TN_p|} = (\frac{rt_{rel(v_{|TN_p|},v_{|TN_p|-1})}}{rt_{rel(v_{|TN_p|-1},v_{|TN_p|-2})}})^{\epsilon_k} \times \ldots \times (\frac{rt_{rel(v_2,v_1)}}{rt_{AC}})^{\epsilon_2} \mod p \quad (3)$$

Due to the property of the $\sigma()$ function in Equation 1, the value of $\sigma_{|TN_p|}$ is 1 (when $\pi = 1$), if all the edge relationship types $rt_{rel(v_j,v_{j-1})}$ for $2 \leq j \leq |TN_p|$ are the same as rt_{AC}, otherwise the value is a uniformly random number in $\{0, \ldots, p-1\}$. Based on the above principle, if $E_{|TN_p|}$ decrypts to 1 (when $\pi = 1$) then the owner concludes that the path relationship type is rt_{AC}, otherwise the path relationship type is not rt_{AC}. We refer the readers to [19] for an example of relationship type computation.

The corresponding token is therefore defined as follows.

Definition 4: **Relationship type token.** Let $p \in DPaths$ be a path discovered during P^3D. Let AC be the access condition considered by v_o in P^3D. For any $v_j \in TN_p$, the relationship type token generated by v_j and denoted as rt_tk_j consists of two components, C_j and E_j, defined as follows:

* if $j = 0$: $C_0 = (rt_{AC} \cdot v_o.pk^{r_0}, g^{r_0})$, $E_0 = (\pi \cdot v_o.pk^{r'_0}, g^{r'_0})$, where r'_0, $r_0 \in \{0, \ldots q-1\}$ are random numbers generated by v_o and π is a specific integer number chosen by v_o.

* if $j = 1$: C_1=C_0; E_1=E_0.

* if $1 < j < |TN_p|$: let $rel(v_j, v_{j-1})$ be the relationship according to which node $v_{j-1} \in TN_p$ sent to $v_j \in TN_p$ rt_tk_{j-1}: $C_j = (rt_{rel(v_j,v_{j-1})} \cdot v_o.pk^{r_j}, g^{r_j})$, where $r_j \in \{0, \ldots q-1\}$ is a random number generated by v_j; $E_j = C_{j/(j-1)}^{\epsilon_j} \cdot E_{j-1} =$ $= ((\frac{rt_{rel(v_j,v_{j-1})}}{rt_{rel(v_{j-1},v_{j-2})}})^{\epsilon_j} \times (\frac{rt_{rel(v_{j-1},v_{j-2})}}{rt_{rel(v_{j-2},v_{j-3})}})^{\epsilon_{j-1}} \times \ldots \times (\frac{rt_{rel(v_2,v_1)}}{rt_{AC}})^{\epsilon_2} \cdot \pi \cdot v_o.pk^r, g^r)$ where $C_{j/j-1} = \frac{C_j}{C_{j-1}}$, and ϵ_j is a random number generated by v_j.

Note that in the above definition, π is initialized by v_o in E_0. Hence, if the path has relationship type rt_{AC}, the decrypted value of $E_{|TN_p|}$, i.e. σ, is π.

V. Handling offline nodes

In this section, we propose an improvement to basic P^3D with the aim of reducing as much as possible the effects of

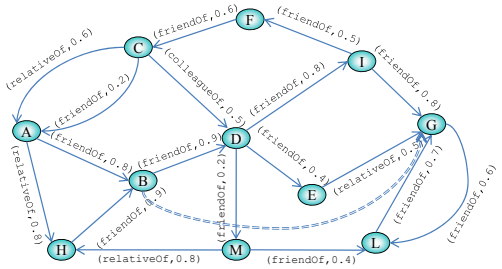


Fig. 1. An example of SN graph with virtual edges.

offline nodes in terms of number of discovered paths. The basic idea is to reuse aggregate information of previously discovered paths. Therefore, at the end of each P^3D execution, for any discovered path $p \in DPath$ the aggregate information extracted from the corresponding tokens is made available for future P^3D executions. This information is modeled as a *virtual edge* connecting the owner with the requestor. When a node is not available, the discovery process can traverse virtual edges, if any, representing the pre-discovered paths to which the offline node belongs to.

Example 1: Consider Figure 1 and suppose that during a P^3D execution where node G required a resource to node B, only the path $p_0 = B \rightarrow D \rightarrow I \rightarrow G$ has been found. Let us assume that a new P^3D execution is started by nodes A and L, as owner and requestor, respectively. P^3D could potentially find 6 paths, all passing through node D.Suppose that node D is offline, P^3D will not find any path. To avoid this, we reuse aggregate information on path p_0, such to jump the offline node. This is possible by allowing G to "traverse" the virtual edge representing p_0 (see Figure 1), such to reach node B and continue the SN traversal.

Note that, according to the path traversal direction, the only node that can traverse a virtual edge is the node that has no information on it (e.g., node G in Example 1). Indeed, in the basic P^3D process only the owner (e.g., B) is able to access aggregate information on the discovered paths, whereas the requestor (e.g., G) only knows that a path connecting it with the owner exists. However, each node to which a virtual edge enters has to be able to "traverse" it. To overcome this limitation we introduce the notion of *virtual edge certificate*.

Definition 5: **Virtual edge certificate.** Let $DPaths$ be the set of paths discovered by P^3D. For any $p \in DPaths$, v_o generates a virtual edge certificate, denoted as $\mathsf{cert}_{VE(v_o,v_r)}$, consisting of: an id, that univocally identifies $\mathsf{cert}_{VE(v_o,v_r)}$;[5] the owner identifier v_o; the time of generation, denoted as t_{gen}; the time of expiration, denoted as t_{exp}; the encryption with the public key of v_o of the depth, relationship type[6] and trust value of p, denoted as $d_{VE(v_o,v_r)}$, $rt_{VE(v_o,v_r)}$, and $tv_{VE(v_o,v_r)}$, respectively, and extracted from the corresponding tokenset received during a past execution of P^3D. As such, $\mathsf{cert}_{VE(v_o,v_r)}$ can be formally defined as: $\mathsf{cert}_{VE(v_o,v_r)} = \langle id, v_o, t_{gen}, t_{exp}, E_{v_o.pk}(d_{VE(v_o,v_r)}||rt_{VE(v_o,v_r)}||tv_{VE(v_o,v_r)})||Sign_{v_o.sk}(id, t_{gen}, t_{exp}, E_{v_o.pk}(d_{VE(v_o,v_r)}||rt_{VE(v_o,v_r)}, ||tv_{VE(v_o,v_r)}))\rangle$; where $E_K()$ and $Sign_K()$ are the encryption and signature with key K, respectively.

Thus, at the end of any P^3D execution, for any discovered path, the owner v_o generates the corresponding virtual edge certificate, which is then sent to the requestor v_r. Both v_o and v_r store the virtual edge certificate into a local repository $\mathcal{CL}$, called *certificate list*. Later on, in another P^3D process, v_r can "traverse" the virtual edge by exploiting $\mathsf{cert}_{VE(v_o,v_r)}$. In particular, it can send to v_o the certificate as a proof of the existence of a path from v_o to v_r, together with the tokenset of the current P^3D process. Once v_o receives $\mathsf{cert}_{VE(v_o,v_r)}$, it can add to the received tokenset the information on the traversed virtual edge, which are retrieved directly from $\mathsf{cert}_{VE(v_o,v_r)}$. As such, virtual edges can be traversed as normal edges. Thus, a simple way to exploit them is that the discovery process traverses both real and virtual edges, with no distinction. However, we should prefer real edges and traverse virtual edges only when no path can be retrieved due to offline nodes, since the aggregated information are fresher than the ones corresponding to virtual edges. Therefore, we have enriched the naive protocol with backtracking, to traverse virtual edges only when needed.

Example 2: Consider the virtual edge in Figure 1 and suppose that: (i) a new discovery process is initialized by A and L as owner and requestor, respectively; (ii) node D is offline. According to the backtracking strategy, L forwards the tokenset initialized by A to G and M. G forwards it to I and E, which propagate the tokenset only to D. Also M forwards the tokenset received from L only to D. Since D is offline, both I and E send a *feedback message* to G, whereas M sends the feedback message to L. Once G receives all the feedback messages, it exploits the virtual edge and reaches B, that is, it sends B the tokenset together with $\mathsf{cert}_{VE(B,G)}$. B is then able to update the received tokenset with information extracted from $\mathsf{cert}_{VE(B,G)}$ and continue the discovery process.

More precisely, once an intermediate node receives a tokenset, it updates each token and then verifies if some non-expired virtual edge certificates are available in its certificate list. If this is the case, the intermediate node randomly selects one of them and sends to its neighbors the updated tokenset together with the id of the virtual edge certificate, which may be potentially exploited in case of offline nodes. To store this information, we introduce a new token, called *virtual edge token*, containing the list of virtual edge certificate ids gathered during the collaborative process. In case of an offline node along the path, the intermediate node that verifies the node unavailability first checks if a non-expired certificate is available in its certificate list. If this is the case, it forwards the tokenset and the certificate id directly to the node representing v_o in the certificate. If no virtual edges are available, the intermediate node sends a *feedback message* back, together with the virtual edge token, to those nodes from which it has previously received the tokenset. When an intermediate node receives a feedback message, it verifies if the virtual edge token contains the hash of a certificate belonging to its certificate list. If this is the case, it traverses the corresponding

[5]We assume that the id is computed as the hash value of the certificate itself, where the id field is set to NULL.

[6]In case the relationship token is π, $rt_{VE(v_o,v_r)}$ is set as the one specified in the access condition, NULL otherwise.

virtual edge. It sends back the feedback message and the virtual edge token, otherwise.

Example 3: Consider the SN in Figure 1 and the backtracking in Example 2. The unique virtual edge certificate available is the one referring to p_0, which has been created by node B and stored into the certificate list of nodes B and G. When L propagates the tokenset to G, the virtual edge token is empty, as L does not have any virtual edge certificate. In contrast, before G forwards the tokenset to I and E, it inserts into the virtual edge token the id of $\mathsf{cert}_{VE(B,G)}$. When I (i.e., E) verifies that D is offline, it first searches if its certificate list contains some certificates, since this is not the case, it sends G a feedback message together with the virtual edge token. Thus, G is able to verify that this token contains the id of its certificate, it retrieves the certificate and sends to B the virtual edge certificate together with the tokenset.

VI. P^3D OPTIMIZATIONS

P^3D is based on message flooding, which may imply high overhead on the network. For this reason, we apply three optimizations to basic P^3D. The first consists of broadcasting the received tokenset only through incoming edges having the relationship type equal to the one of the edge from which the tokenset has been received. Note that this optimization does not impact the number of indirect relationships found by P^3D, as we exclude only paths having edges with different relationship types. Note moreover that, the side-effect of this optimization is that a node, say v, can infer that the node from which it received the tokenset has an exiting edge of the same relationship type of the one between them, without knowing with whom. However, as it will be discussed in Section VII (cfr. Theorem 7.2), this information gain is intrinsic in the proposed discovery protocol, as each node could play the owner role during a P^3D execution.

The second optimization aims to reduce tokenset propagation by stopping redundant ones. If a node v receives two tokenset, say TS_1 and TS_2, from the same node v', and TS_1 contains a trust value greater than TS_2, only TS_1 is propagated. Indeed, since TS_1 contains a trust value greater than TS_2, it is unnecessary to forward TS_2 to the next hops as if the path traversed by TS_2 satisfies the trust constraint in the access condition defined by the owner then the path traversed by TS_1 must also satisfy it. Similarly, if TS_1 contains a depth value smaller than TS_2, it is unnecessary to forward TS_2 to the next hops. When we consider trust and depth together, TS_2 is considered redundant if its depth is not smaller than that of TS_1 and its trust value is not larger than that of TS_1. To apply such optimization, we have to modify P^3D such that: (1) nodes are able to compare the trust and the depth values in different tokensets; (2) to propagate fewer tokensets as possible, the tokenset with greater trust and smaller depth should be received by the nodes earlier so that other tokensets with lower trust value and larger depth value can be discarded by the nodes. Fortunately, our protocol can be modified to realize both (1) and (2). Since we use hashing to encode the trust and depth values, even if intermediate nodes are not able to access the real values contained in the tokens, they can compare their values to find the greatest or smallest one. Second, to make tokensets with good trust and depth values arrive at next hops earlier, we introduce the *tokenset delay* mechanism. This imposes to apply a delay before propagating the tokenset, where the delay is determined based on trust value of the edge from which the tokenset has been received: the greater the edge's trust value is, the shorter is the delay. Note this optimization the protocol may drop the paths that are less optimal, but never omits paths that are potentially optimal. Hence it is applicable to access control policies that evaluate relationship and trust based on optimal paths.

The third optimization limits the SN traversal to the maximum depth required in the access condition. For example, if the access condition AC requires a relationship with a maximum depth of 5, it is a waste of time searching paths with length greater than 5. As such, passing the information on $depth_{AC}$ during P^3D would optimize the cost of the discovery process. However, as we aim to protect also the privacy of access conditions, we do not pass the real $depth_{AC}$, rather an upper bound of it, generated by adding to $depth_{AC}$ a random number r. Therefore, an additional parameter is passed together with the tokenset, called $maxDepth$, initialized as $depth_{AC}+r$, which each intermediate node decrements by one before propagating to the next nodes. The $maxDepth$ parameter also serves another function for the protocol: it guarantees the message propagations to be stopped eventually, and hence the path discovery protocol process always terminates.

The full algorithm descriptions and pseudo-codes for the basic protocol, offline nodes handling and optimization can be found in [19].

VII. SECURITY ANALYSIS

We assume a semi-honest adversary model, such that the adversary nodes (i.e., any intermediate node) follow the execution of the protocol, but are curious in gaining additional information based on the set of received messages/tokensets and their background knowledge. As background knowledge, we assume that each node knows any property (i.e., trust and relationship type) of *exiting edges*, whereas it only knows the relationship type of *incoming edges*, since, according to the proposed model, the trust value is known only by the node who established the relationship. The assumption of the semi-honest model follows previous work on privacy-preserving data mining (e.g., for path discovery in social networks [16], for decision tree computation [14], or for privacy-preserving classification [20].) Indeed, as observed in [9], the semi-honest model fits well for those data mining applications where involved entities are the data holders their selves, which in general refrain from illegal actions since they are interested only in following protocol steps. The following theorem states the privacy guarantees of P^3D.

Theorem 7.1: Let AC be the access condition applying on the requested resource. For any $p \in DPaths$, the following properties hold: (P_1) only v_o is aware of the trust, relationship type and depth specified in AC; (P_2) only v_o is aware of the number of edges in p; (P_3) only v_o is aware of the trust of p; (P_4) only v_o is aware of the relationship type of p. In particular,

the owner is able to verify only that the relationship type of all the edges in the path, i.e., $rt_{rel(v_j,v_{j-1})}$, $j = \{2,\ldots,|TN_p| - 1\}$ is equal or not to the one specified in AC; (P_5) the identity of v_r is kept private to all intermediate nodes, and to v_o, if v_r is not a direct contact of v_o; (P_6) the identity of intermediate nodes is hidden to everyone, apart from direct contacts.

It is important to note that even if, according to Properties 2,3,4 the owner is able to know the depth, relationship type and trust value of the discovered path, it does not know the identity of the requestor (P_5), nor of any intermediate node, apart from its contacts (P_6). Thus, the owner knows the existence of an indirect relationship with certain properties but it does not know with whom it has this relationship. Graphically, this can be modelled as a pending path exiting from the owner and passing through the owner's direct contact from which it received the tokenset corresponding to that path. We refer to this information gain as a knowledge on *neighbors exiting pending paths*.

We have to note, however, that this information gain is intrinsic in the problem we are addressing. Indeed, according to the proposed access control model, the depth, relationship type and trust value of an indirect relationship with the requestor is the minimum information needed to take the access control decision. Keeping hidden the identity of the requestor reduces the knowledge about this indirect relationship. As such, the proposed path discovery process makes the owner able to enforce its access conditions with the minimum privacy leakage. Note that, a particular case is when the path has depth=2. In this case, the owner is able to infer that the node from which it received the tokenset (i.e., one of its direct contacts, say node X) has a relationship with a given trust and relationship type with the requestor. However, as the owner does not know the requestor identity, this can be still considered as a pending edge exiting from X. Thus, the worst case is obtained when the owner receives from each of its neighbors a tokenset identifying a path of length 2. Indeed, in this case the owner is able to infer the properties of a pending edge (i.e., trust and relationship type) exiting from each of its neighbors. However, as P^3D keeps the requestor identity hidden, even in the worst case the owner is not able to learn any additional information about the graph, besides the above discussed one. The following theorem shows that the knowledge on *neighbors exiting pending paths* is the maximum knowledge any node in the SN can gain by using the information gathered during the execution of P^3D.

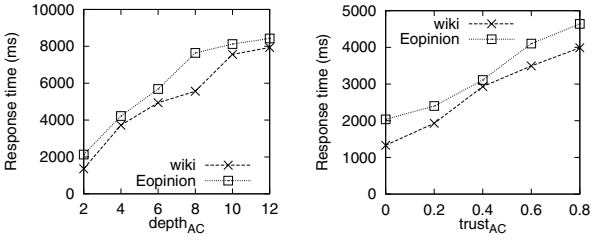
Theorem 7.2: Let $V \in V_{SN}$ be a social network node, and let IG be the information gained during past P^3D executions where V has been involved. IG is less than or equal to the knowledge on *neighbors exiting pending paths* inferred by V if it played the owner role.

The formal proofs of Theorems 7.1 7.2 are reported in [19].

VIII. Experimental Evaluation

In this section, we first evaluate the performance of P^3D, by varying the access condition complexity, the number of offline nodes and the number of virtual edges. In the first set of experiments, two main measures have been considered to show P^3D feasibility. The first is the *response time*, that is, the time needed to discover the first path satisfying the considered access condition, if any, or to explore all possible paths, otherwise. The second is the *grant ratio*, that is, the percentage of access requests receiving a positive answer over the total amount of requests. These measures have been computed by performing each experiment 200 times, so to have 200 different pairs of owners and requestors. Last, we compare our approach against the one presented in [6], in particular w.r.t efficiency of the trust value computation. To evaluate P^3D performance in a more realistic network environment, the prototype has been implemented on top of the Java Network Simulator (JNS),[7] which provides APIs for network layer simulation. Specifically, using JNS we establish a reliable duplex data link with 50kbps data rate (easily achievable in 3G networks) for each pair of communicating nodes for exchanging data packets. All experiments were run on Windows XP machine with 4Gb Ram and 2.53GHz duo CPU.

Datasets. All experiments have been conducted on real SN data. In particular, we have considered: Wiki-vote and Eopinion SNs[8] with 7,115 nodes and 103,689 edges, and 75,879 nodes and 508,837 edges, respectively. Since our SN model supports trust values and multiple relationship types, we have modified both datasets such to randomly assign trust values and relationship types to each edge: (1) the trust value is randomly selected from a normal distribution with 95% confidence in the interval [0,1];[9] the relationship type is uniformly and randomly picked up from a set of relationship types RT_{exp}, whose size varies in the experiments. Each experiment run requires a pair of requestor and owner nodes, which are selected according to the following steps: (1) a random requestor v_r is selected among all the nodes in the dataset; (2) from the nodes that have a direct relationship (either in-coming or out-going) with the previously selected node, a further node is picked up; (3) step 2 is repeated l times; (4) the l-th selected node plays the owner role. As such, we define a random walk of l edges-length from the requestor to the owner, where in our experiments l is set to 6.



(a) Time vs. $depth_{AC}$ (b) Time vs. $trust_{AC}$

Fig. 2. Access condition complexity

Varying access condition complexity. In this set of experiments, we vary the complexity of access conditions, that is, $depth_{AC}$ and $trust_{AC}$.

[7]http://jns.sourceforge.net.

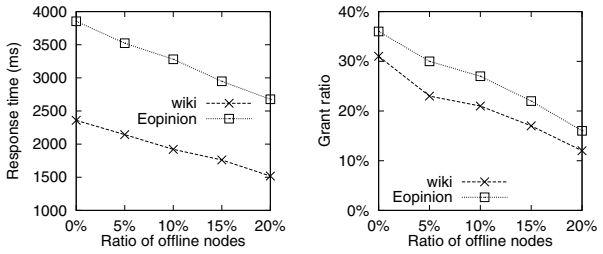
[8]Both available at http://snap.stanford.edu/data/.

[9]The random selection is repeated until the selected value falls into [0,1].

To evaluate the performance with respect to $depth_{AC}$, we set $trust_{AC} = 0.7$, and randomly select rt_{AC} in RT_{exp}, where $|RT_{exp}| = 4$ and RT_{exp} denotes the set of relationship types avaialable for experiments. $trust_{AC}$ has been fixed to 0.7 as this represents a quite strict constraint, so to keep high the access condition complexity. $|RT_{exp}|$ has been fixed to 4 to increase the availability of indirect relationships in the considered datasets.

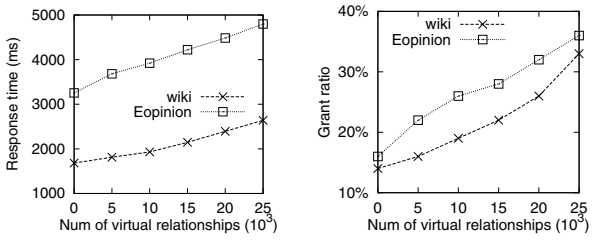
Figure 2(a) shows how the response time changes when we vary $depth_{AC}$ from 2 to 12. For both datasets, the response time increases with the $depth_{AC}$ value. To evaluate access condition complexity w.r.t. trust, we fix $depth_{AC}$ = $|RT_{exp}| = 4$ and vary $trust_{AC}$ from 0 to 0.8. We have fixed $depth_{AC} = 4$ because according to the six-degree small world property [17] each pair of nodes is connected within six-length path. As such, setting $depth_{AC} = 4$ gives us a depth condition that is not satisfied by each pair of owner/requestor. Figure 2(b) shows that the response time increases when we increase $trust_{AC}$. This is because higher $trust_{AC}$ imply more strict access conditions, so less paths satisfying AC.

We have also evaluated the performance of P^3D by fixing $depth_{AC}$ and $trust_{AC}$, and varying the size of RT_{exp} from 1 to 8. Results can be found in [19].



(a) Time vs. ratio offline (b) Grant vs. ratio offline

Fig. 3. Ratio of offline nodes



(a) Time vs. num. virtual (b) Grant vs. num. virtual

Fig. 4. Number of virtual edges

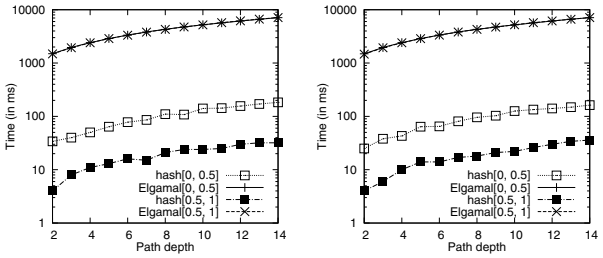
Varying ratio of offline nodes. The second set of experiments aims to show how the presence of offline nodes impact both response time and grant ratio. We fix $depth_{AC} = 4$, $trust_{AC} = 0.7$, $|RT_{exp}| = 4$, and assume no virtual edge. Figure 3(a) shows how the response time decreases when increasing the percentage of offline nodes. Moreover, as the increase of the percentage of offline nodes implies that less paths between the requestor and the owner are available, it is expected a decrease of the grant ratio as shown in Figures 3(b).

Varying the number of virtual edges. In the last set of experiments, we evaluate the effectiveness of using virtual edges to improve path availability. Here, we set 20% percent of randomly selected nodes as offline, and increase the number of virtual edges from 0 to 25,000. In particular, to insert a fixed number of virtual edges into the network, we select a fixed number of requestors and owners (i.e., from 0 to 25,000) by using the same strategy adopted for owner and requestor selection. Then depth, trust value, and the relationship type of the path with the best trust value between the requestor and the owner is used to generate the virtual edge. Figure 4(a) shows that when we increase the number of virtual edges, the response time also increases. We recall that, in case of offline nodes, P^3D backtracks until it finds a virtual edge, if any, to traverse. As such, increasing the number of virtual edges has the effect of increasing the network connectivity, i.e., the number of edges to be traversed. However, Figure 4(b) shows that the increase of the number of virtual edges improves the grant ratio in the network. As expected the more virtual edges are available the higher is the probability to find a path matching the access condition. The experiment shows that virtual edges help to overcome the offline nodes problem.

Comparison in trust value aggregation. As discussed in Section I, our protocol ensures new security properties. However, despite the differences with other protocols, a common feature of all these protocols is the secure trust values computation. As such, we focus on this computation to show comparison existing solutions. In particular, we are interested in comparing our approach with the one presented in [6]. We choose this work since it exploits a different technique, i.e., ElGamal encryption, w.r.t. the one exploited in this paper, i.e., hash chain (ref. Section IV-B).

In our experiments, we compare the time needed for aggregating the trust values of paths of various lengths. Specifically, we create paths of lengths from 2 to 14. For each edge in the path, we assign a trust value from uniform distribution or normal distribution in the interval $[0, 0.5]$ or $[0.5, 1]$. We use standard normal distribution, and map the value within the 95% confidence range into the interval $[0, 0.5]$ or $[0.5, 1]$. The rationale of dividing the trust values into 2 intervals is that the performance of our approach relies on the trust values as these determine the number of hashing iterations to be performed. The hash algorithm that we use is SHA-512 which maps a message into a 512 bits long digest. To fairly compare with ElGamal encryption, we also set the key length of ElGamal encryption to 512 bits length. Figure 5(a) and (b) show the efficiency comparison of trust aggregation under uniform and normal distribution, respectively. Both two figures show very similar pattern and they clearly show that our approach (hash) is much more efficient compared to the Elgmal encryption (ElGamal) based approach. This result is as expected because hashing is generally much more efficient than public key based encryption (i.e., the ElGamal encryption). While our result is consistently better, the time needed for aggregating lower trust values (i.e. sampled from the interval $[0, 0.5]$) is higher than the time needed to aggregate higher trust values (i.e., sampled from $[0.5, .1]$). This observation matches our expectation as our algorithm takes more hashing iterations to aggregate a lower trust value, and hence resulting longer time. We also notice the intervals from which the trust values are taken has

no noticeable impact over the performance. This is because the ElGamal encryption takes almost fixed rounds of operations regardless the trust values. From the experiments, we conclude that our hash based trust aggregation, while offering additional security guarantees, is extremely efficient and outperforms the ElGamal encryption based approach.



(a) Uniform distribution (b) Normal distribution

Fig. 5. Efficiency comparison of trust aggregation

Discussion. As expected the access condition complexity impacts both response time and ratio of grant. Indeed, more strict conditions in terms imply always an increase of response time and a decrease of grant ratio. However, even with complex access conditions, the response time is always feasible (i.e., less than 10sec in the worst case, see Figure 2(a)). The second set of experiments highlight that path discovery protocols suffer from the presence of offline nodes. As shown in Figure 3(b), the worsening of the grant ratio is fast (from more than 30% with all nodes online to less than 20% with 20% of offline nodes). This confirms us that a path discovery process has to be enhanced with strategies able to deal with the presence of offline node. Finally, the third set of experiments show the effectiveness of the proposed solution to handle offline nodes. As shown in Figure 4(b), the increasing number of available virtual edges improves the grant ratio, which starts from less than 20% of grants when no virtual edges is available and reaches more than 30% with 25,000 virtual edges. Note that this is the same percentage obtained when all nodes are online (see Figure 3(b)). As such this experiment shows that the virtual edges fully overcome the problem of offline nodes. Finally, we compared the proposed approach for trust computation with the one presented in [6]. These experiments show that our approach is much more efficient than encryption-based ones.

IX. Conclusion

In this paper, we have presented P^3D, a privacy-preserving path discovery protocol for SNs. P^3D protects the privacy of relationship by, at the same time, making it possible the enforcement of resource releasing rules, based on path depth, type and trust. Other relevant features of P^3D are that it does not rely on a centralized SN architecture, with obvious privacy benefits, and that it is equipped with techniques to reduce the effects of offline nodes on path discovery. We plan to extend this work along several directions, such as for instance, the efficient support for different trust computation algorithm, where the flooding optimization strategies introduced in Section VI could be adapted to different way of computing trust. We also plan to devise strategies to properly set the time of expiration of virtual edge certificates, such that they can immediately catch any update on the social graph.

Acknowledgement

The research presented in this article was partially funded by a Google Research Award and by the Mi-Fido project, sponsored by the Italian Ministry of Economic Development.

References

[1] B. Carminati, E. Ferrari, and A. Perego. A decentralized security framework for web-based social networks. *International Journal of Information Security and Privacy*, 2(4):22–53, 2008.

[2] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM Trans. Inf. Syst. Secur.*, 13(1):1–38, 2009.

[3] K. S. R. G.-S. S. K. D. B. Choi, H.-C. and J. G. Breslin. Trust models for community-aware identity management. In *In Identity, Reference, and the Web Workshop, IRW*, 2006.

[4] L. A. Cutillo, R. Molva, and T. Strufe. Privacy preserving social networking through decentralization. In *WONS'09*, pages 133–140, 2009.

[5] J. Domingo-Ferrer. A public-key protocol for social networks with private relationships. In *MDAI '07*, pages 373–379, 2007.

[6] J. Domingo-Ferrer, A. Viejo, F. Sebé, and ı. González-Nicolás. Privacy homomorphisms for social networks with private relationships. *Computer Networks*, 52(15):3007–3016, 2008.

[7] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, 1985.

[8] K. B. Frikken and P. Srinivas. Key allocation schemes for private social networks. In *WPES '09*, pages 11–20, 2009.

[9] F. Giannotti and D. Pedreschi, editors. *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*. Springer, 2008.

[10] J. A. Golbeck. *Computing and applying trust in web-based social networks*. PhD thesis, 2005. Chair-Hendler, James.

[11] D. N. Kalofonos, Z. Antoniou, F. D. Reynolds, M. Van-kleek, J. Strauss, and P. Wisner. Mynet: a platform for secure p2p personal and social networking services. In *PerCom'08*, 2008.

[12] H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *Protocol, Advances in Cryptology CRYPTO 05, LNCS 3621*, pages 546–566. Springer-Verlag, 2005.

[13] G. S. C. H.-C. W. T. Kruk, S. R. and A. Gzella. D-foaf: Distributed identity management with access rights delegation. In *The Semantic Web ASWC 2006. LNCS, vol. 4185. Springer*, pages 140 – 154, 2006.

[14] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3), 2002.

[15] A. P. McAfee. Enterprise 2.0: The dawn of emergent collaboration. *MITSloan Management Review*, 47(3):21–28, 2006.

[16] G. Mezzour, A. Perrig, V. Gligor, and P. Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *CANS '09*, pages 189–208, 2009.

[17] M. Newman, A.-L. Barabasi, and D. J. Watts. *The Structure and Dynamics of Networks: (Princeton Studies in Complexity)*. Princeton University Press, 2006.

[18] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam. Prpl: a decentralized social networking infrastructure. In *MCS '10*, pages 8:1–8:8, 2010.

[19] M. Xue, B. Carminati, and E. Ferrari. p^3d - privacy-preserving path discovery in decentralized online social networks. *Technical report, url:*`http://www.comp.nus.edu.sg/~xuemingq/p3d.pdf`.

[20] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *In Proceedings of Fifth SIAM International Conference on Data Mining*, 2005.