Secure information sharing on Decentralized Social Networks

University of Insubria

Department of Theoretical and Applied Sciences

Davide Alberto Albertini

Supervisor:
Prof.ssa Barbara Carminati

External Reviewers:
Prof. George Pallis
Dr. Sergio Mascetti

This dissertation is submitted for the degree of
Doctor of Philosophy in Computer Science
17th March 2017

Ai miei genitori, a Elisa e a tutta la mia famiglia,
grazie ai quali sono la persona che sono
e ai quali posso dire solo un sincero grazie.

Acknowledgements

Say thank to someone after such a long journey is never easy; the most of the times it happens that the people who have been really important stay unmentioned only because of the amount of time had together made their presence implied.

For this reason I believe that the first mention should be to my family and to all my dearest. Parents, relatives, friends, acquaintances, and any person from which I learnt even a slight aspect of my behaviour deserve to be mentioned, since their presence in my life made me the man that I am now.
Among these, I will always be grateful to my Ph.D supervisor, *Dr. Barbara Carminati*, who patiently pushed my career with honesty and candour. I may have not been the best of your students, but I will always bring your teachings with me.
As well, I must also thank all the people with whom I spent most of the last four years in Varese. *Alberto, Mauro, and Pietro (and anyone else who I am forgotting)*, your company and our cheerful conversations made valuable every day spent at the first floor.

A special and particular acknowledgement has to be given to *Elisa*, my girlfriend, who always tried to make the best out of me, to *Giuseppe* and *Milena*, my parents, who have been source of unconditional support in all of my life, and to *Ljuba*, my sister, who -*somehow*- inspired me in pursuing my dreams with hers stubbornness and hers obstinacy.

To all of you, I will never be able to thank you enough.

Abstract

Decentralized Social Networks (DSNs) are web-based platforms built on distributed systems (federations) composed of multiple providers (pods) that run the same social networking service. DSNs have been presented as a valid alternative to Online Social Networks (OSNs), replacing the centralized paradigm of OSNs with a decentralized distribution of the features offered by the social networking platform.
Similarly to commercial OSNs, DSNs offer to their subscribed users a number of distinctive features, such as the possibility to share resources with other subscribed users or the possibility to establish virtual relationships with other DSN users. On the other hand, each DSN user takes part in the service, choosing to store personal data on his/her own trusted provider inside the federation or to deploy his/her own provider on a private machine. This, thus, gives each DSN user direct control of his/hers data and prevents the social network provider from performing data mining analysis over these information.

Unfortunately, the deployment of a personal DSN pod is not as simple as it sounds. Indeed, each pod's owner has to maintain the security, integrity, and reliability of all the data stored in that provider. Furthermore, given the amount of data produced each day in a social network service, it is reasonable to assume that the majority of users cannot afford the upkeep of an hardware capable of handling such amount of information. As a result, it has been shown that most of DSN users prefer to subscribe to an existing provider despite setting up a new one, bringing to an indirect centralization of data that leads DSNs to suffer of the same issues as centralized social network services.

In order to overcome this issue in this thesis we have investigated the possibility for DSN providers to lean on modern cloud-based storage services so as to offer a cloud-based information sharing service. This has required to deal with many challenges.
As such, we have investigated the definition of cryptographic protocols enabling DSN users to securely store their resources in the public cloud, along with the definition of communication protocols ensuring that decryption keys are distributed only to authorized users, that is users that satisfy at least one of the access control policies specified by data owner according to Relationship-based access control model (RelBAC) [20, 34].
In addition, it has emerged that even DSN users have the same difficulties as OSN users in defining RelBAC rules that properly express their attitude towards their own privacy. Indeed, it is nowadays well accepted that the definition of access control policies is an error-prone task. Then, since misconfigured RelBAC policies may lead to harmful data release and may expose the privacy of others as well, we believe that DSN users should

be assisted in the RelBAC policy definition process. At this purpose, we have designed a RelBAC policy recommendation system such that it can learn from DSN users their own attitude towards privacy, and exploits all the learned data to assist DSN users in the definition of RelBAC policies by suggesting customized privacy rules.

Nevertheless, despite the presence of the above mentioned policy recommender, it is reasonable to assume that misconfigured RelBAC rules may appear in the system. However, rather than considering all misconfigured policies as leading to potentially harmful situations, we have considered that they might even lead to an exacerbated data restriction that brings to a loss of utility to DSN users. As an example, assuming that a low resolution and an high resolution version of the same picture are uploaded in the network, we believe that the low-res version should be granted to all those users who are granted to access the hi-res version, even though, due to a misconfiurated system, no policy explicitly authorizes them on the low-res picture. As such, we have designed a technique capable of exploiting all the existing data dependencies (i.e., any correlation between data) as a mean for increasing the system utility, that is, the number of queries that can be safely answered. Then, we have defined a query rewriting technique capable of extending defined access control policy authorizations by exploiting data dependencies, in order to authorize unauthorized but inferable data.

In this thesis we present a complete description of the above mentioned proposals, along with the experimental results of the tests that have been carried out so as to verify the feasibility of the presented techniques.

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

In recent years Online Social Networks (OSNs) emerged as one of the most promising platforms for information sharing on the Web, with remarkable impact on people's everyday life.
The variety of currently available OSNs platforms does not allow to give a precise definition of this type of information sharing platforms. Nevertheless, most of the OSNs share a number of common features, such as the possibility to upload and share files, pictures, and any other kind of user-generated content (UGC), or the necessity for OSN users to create a platform-specific account (i.e., a digital identity) so as to access the OSN services. These aspects, like other features common to many commercial OSNs, rely on the existence of a centralized entity entity managing all users' information (e.g., UGCs, users' real identities, etc), as depicted in Figure 1.1. This represents the weakest point for OSNs for several reasons. At first, as properly emphasized in [71], companies providing centralized social networking services have the sole authority to manage and control any kind of users' data; despite OSN users are required to approve legal agreements that regulate how their data can be exploited by the OSN provider, this does not give OSN users direct control over their own information.
As an example, even though a given user decides to delete all of his/hers information from the OSN, these data are not removed from the OSN realm but they remain stored in the OSN repositories. Thus, when a certain user chooses to delete something he/she shared on Facebook, the OSN provider removes it from the site. Some of this information is permanently deleted from provider's servers; however, some things can only be

deleted when the user chooses to permanently delete his/hers account[1]. In Twitter, similarly, log data (i.e., usernames, IP addresses, email addresses, phone numbers) are kept by the service provider as much as needed and deleted after a maximum of 18 months. If a certain user chooses to delete his/hers account, the OSN provider may keep all of his/hers data up to one week[2]. Nevertheless, search engines and other third parties may still retain copies of users' public information even after users deleted their own OSN accounts. Indeed, in the most of the cases, the legal ownership of any data changes from the uploading user to the OSN provider as soon as these data are uploaded in the OSN realm, without any possibility for the uploading user to control how these data are treated by the social network provider.
As such, we believe that this lack of control over personal information is an excessive price to be paid from users in return for OSN services subscriptions. Then, we believe that users' privacy in social network realms should be preserved by giving each user complete control over his/hers own data, as stated by Charles Fried in [35]:

"Privacy is not simply an absence of information about us in the minds of others; rather it is the control we have over information about ourselves."

Moreover, beyond privacy issues, data centralization currently implemented by OSN providers often leads to security issues as well. Indeed, storing a massive amount of personal information into a centralized data repository creates, in the OSN architecture, an entity that can be easily target for extracting a large number of information with a relatively low effort. As an example, it is reasonable to assume that a malicious entity (e.g., an expert attacker, or a competitor OSN company) may be interested in extracting data from the profiles repository, that is where the OSN accounts are stored along with the corresponding real people identities. At the same time, the social graph representing all the relationships existing in the OSN and the personal information repository could be an interesting target for a malicious entity too, given the amount of information they gather.
As a countermeasure to protect such information, OSN providers deploy many of the most secure protocols for data protection, such as TLS over HTTP (HTTPS) for encrypted data transmission or OAuth for authorized data release, but it has been proved [39] that such strategy is not effective as it sounds. Indeed, not all the information collected in the OSN repositories can be transmitted to the OSN users as encrypted data, making the transmission channel a weak point in the the OSN architecture. As an example, accessing an OSN by means of a web browser requires that an HTML page is transmitted from the OSN server to the connecting user. This web page, which contains a remarkable amount of information about the connecting user and his/hers contacts, is

[1]https://www.facebook.com/about/privacy
[2]https://twitter.com/privacy?lang=en

solely protected by HTTPS that can be intercepted and broken by malicious entities.

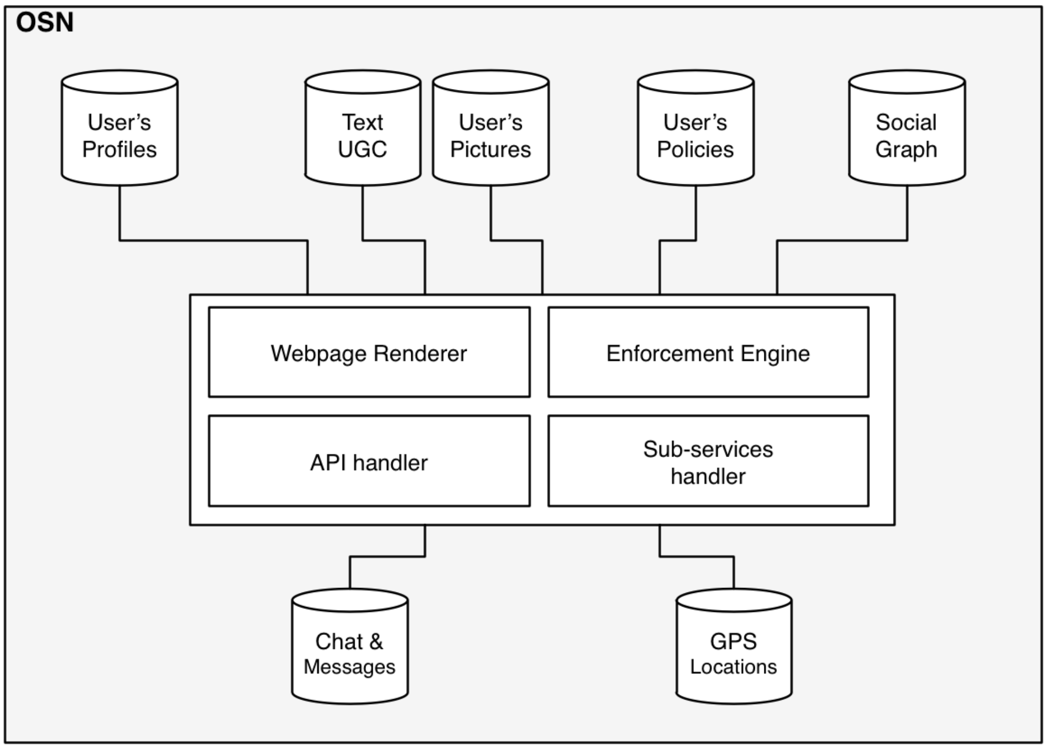


Figure 1.1: OSN reference architecture

However, protecting users' privacy from malicious entities does not prevent OSN providers to exploit the information collected in the OSN repositories disregarding users' privacy. Indeed, as introduced before, in order to implement the above mentioned features OSN providers rely on a centralized architecture, where all users' information is collected. Then, this grants OSN providers a direct access on every user information and enables them to exploit data mining and user profiling techniques so as to monetize users' information by selling such data to advertising or marketing companies.
As such, it emerged the necessity to protect OSN users' privacy, preventing social network providers to exploit their centralized architecture by monetizing from users' personal information. Indeed, as emphasized in [59], techniques like the separation of users' profile data from the social graph or social graph anonymization are not enough to avoid that the OSN manager is able to perform marketing research on users' personal data or access users sensitive information.

Decentralized Online Social Networks (DSNs), also known as Federated Social Network, have been presented in [71] as a valid alternative to OSNs, replacing the centralized paradigm of OSN with a decentralized distribution of the features offered by the social networking platform.

1.1 Motivations

As highlighted by Tim Berners-Lee *et al.* in [71], DSNs emerged as promising solution for moving users' personal data out from OSN realms. A DSN consists in a distributed system (federation) composed of multiple providers (pods) that run the same social networking service. In a DSN, each subscribed user takes part in the service, choosing to store personal data on his/her own trusted provider inside the federation or to deploy his/her own provider on a private machine. Notable examples of federated social networks are Diaspora [2, 11] or OneSocialWeb [3], whereas other examples can be found in Persona [6], SafeBook [24, 23], PeerSoN [16], Vegas [29], Vis-à-Vis [61] and DECENT [44]. Figure 1.2 depicts a DSN reference model.

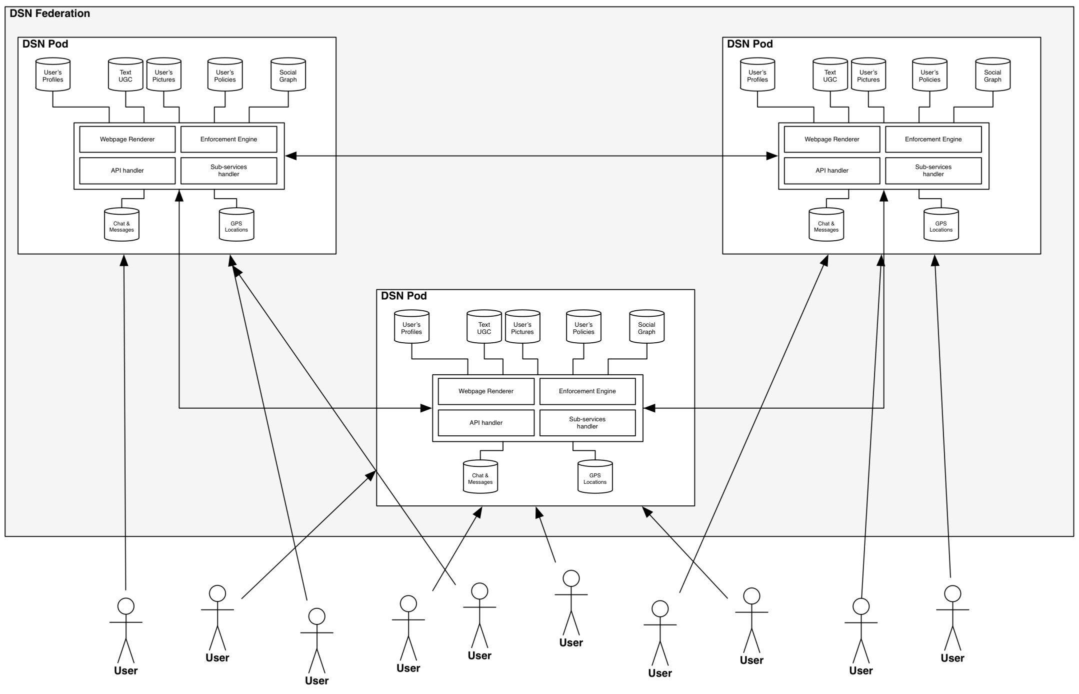


Figure 1.2: DSN reference architecture

Inspired by authors in [71] and their presentation of the decentralized paradigm, DSNs have been the main research topic of all the proposals gathered in this thesis. By analyzing real DSNs implementation we noticed that the most of them suffer of precise and significant issues that limit their potentialities. As such, our researches have been

conducted so as to tackle these issues in order to provide novel technologies for DSN providers.
As an example, it has been shown that few DSN users are able to set up their own provider. This brings to an indirect centralization of data, since users prefer to subscribe to an existing provider despite setting up a new one (e.g., the 70% of Diaspora users are subscribed at the same provider [11]). As such, this leads DSNs to suffer of the same issues highlighted before for centralized social network services.
In addition to this, given the amount of data produced each day in a social network service, it is reasonable to assume that the majority of users cannot afford the upkeep of an hardware capable of handling such amount of information. Indeed, as an example, in 2014 Facebook claimed to produce up to 600 TB of data each day, with a data warehouse capable of storing up to 300 PB of data.[3] Then, it emerged the necessity to implement new technologies enabling DSN architecture to lay on modern cloud-based storage services (e.g., Dropbox, Google Drive, OneDrive, etc.) in order not to burden on DSN users'. Although, this feature is not as simple as it sounds. Indeed, enforcing access control rules in a cloud-based DSN leads to deal with many challenges.

As a first challenge to be dealt with, we believe that the design of a cloud-based DSN should have to follow the same principles that regulates the information sharing in actual OSNs. In particular, OSN providers let their users specify their own access control rules (i.e., the principles according to which a resource should be released or not) by means of simple and intuitive tools. It is nowadays well accepted that in this domain information is released according to relationship-based access control model (RelBAC) [20, 34]. In general, by means of RelBAC rules, information owner can pose conditions on the relationship(s) (e.g., on its type, depth, and possible trust value) the requestor has to fulfill in order to be authorized to access his/her resources. As a result, then, OSN providers are required to release information only to those requestor users that satisfy the constraints defined by the information owner, ensuring information owner that no data will be released to those requestors that do not satisfy any rule.
Another remarkable challenge that have to be tackled is that a cloud-based DSN platform has to provide cryptographic protocols suitable for the considered scenario, in that users' data should be encrypted prior to be stored in the cloud but, at the same time, only those users who fulfill access control rules should be able to decrypt them. Then, this requires to release the decryption key of a certain resource only to those DSN users who fulfill the constraints posed in the access control rule defined for that considered resource.
Additionally, as mentioned before, verifying an access control rule in a DSN scenario

[3] https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

requires to check the existence of a connection between two given nodes of the social graph. However, this verification should be carried out preserving both users' privacy and the privacy of existing relationships. Thus, despite users' data can be encrypted in order to preserve users' privacy, this is not a feasible approach for what concerns the social graph. Nevertheless, social graph anonymization techniques represent a valuable alternative. Current literature offers several techniques for graph anonymization. However, most of the presented techniques anonymize the graph by either clustering nodes (e.g., [10, 17, 42]) or modifying the number of edges in the graph (e.g., [31, 43, 48]), and this is not a feasible approach for enforcing RelBAC policies. Moreover, the majority of the graph anonymization proposal have been designed under the assumption that the anonymization process is carried out on the whole graph. This approach, thus, requires to memorize the social graph in a centralized entity and, as mentioned before, this does not protect users' privacy since it leads to situations in which users do not have direct control over their own information. As such, it emerges the necessity to define a novel privacy-preserving anonymization technique for the social graph that leaves the possibility to verify the existence of a connection between two given nodes without disclosing any users' information, and giving users direct control over their own relationships data. As a result, implementing a privacy preserving information sharing in a cloud-based DSN scenario requires to deal with all the above mentioned challenges.

In addition to the above illustrated issues, it emerged that social network users have difficulties in defining RelBAC rules that properly express their attitude towards their own privacy, mainly for two reasons.

The first reason is due to the fact that many of the most popular OSN providers do not implement in their platforms access control models fully compliant with RelBAC, as properly emphasized in [19]. In some cases fine-grained configurations are supported by giving users the possibility to define customized lists of contacts to which assign access privileges. In other cases, though, users are only asked to choice whether a resource should be *private* or *public* (i.e., released to anyone in the social network). This fact, thus, limits the users' capability to define access control policies such that these reflect their own attitude towards privacy.
Moreover, average users might have difficulties in properly setting, potentially, complex access control policies. As results, as underlined in [39, 50], it emerged that users do not put much attention on customizing RelBAC rules, delegating to the social network provider the task to handle automatically their configurations by exploiting pre-defined policies. As such, given the above mentioned issues, it emerges the necessity to investigate the definition of tools that help users in configuring their own privacy settings.

The second reason is given by the fact that, sometimes, users misconfigure their own RelBAC rules, leading to potentially harmful situations. Indeed, it has been shown

that misconfigured access control policies may lead to sensitive data release. Then, in a scenario where always more frequently personal and sensitive information is exchanged among users, this could even cause to expose other users' privacy. As an example, if a certain users shares a picture setting erroneously its RelBAC policy as public, then even the privacy of every person depicted in such picture would be exposed.
On the other hand, though, misconfigured access control policies may even lead to an exacerbated data restriction that brings to a loss of utility to the OSN users. As such, let assume that a certain OSN users authorizes a precise set of users to see the value of his/hers birth date, but he/she forgets to extends the mentioned authorized *"age"* value. Then, according to RelBAC model, none of the above mentioned users would be able to access the value of the *"age"* field, even though this information is implicitly released by releasing the birth date. In general, thus, it emerged the necessity to define techniques with the aim of handling misconfigured RelBAC policies.

As such, we believe that new proposals should be presented to offer OSN users valid alternatives to centralized architectures, by exploiting the potential of modern cloud-based storage services. At the same time, though, we think that even DSN providers require new and more flexible strategies helping them in the management of RelBAC policies.
For the above described reasons, then, we found it necessary to improve the DSN architecture, offering novel and secure technologies that lead DSN users to a more aware and self-conscious usage of DSN platforms.

1.2 Objective

As introduced in previous Section 1.1, we are aware that current DSN providers suffer of precise limitations. Whereas some of these are due to the decentralized architecture exploited by service providers, other limitations are due to the implementation of the RelBAC model.
Inspired by Charles Fried's statement we tried to pursue the definition of novel technologies for DSN providers enabling their users to have a more safe and complete control over the information that they share in the social network realm.

As such, with regard to the previously described problems, in the following we highlight the topics that we consider well-worthy of further studies.

Cloud-based secure information sharing. In order to enable current DSN architecture to lay on modern cloud-based storage services, we believe that the DSN architecture, as depicted in Figure 1.2, needs to be extended by introducing new entities. These entities, acting as a bridge between the current DSN architecture and cloud-based storage

services, should implement cryptographic primitives to encipher users' resources.
Along with the introduction of new entities in the architecture, DSN providers require the definition of an ad-hoc communication protocol ensuring that resources decryption keys are delivered only to those users who are effectively authorized to access the considered resource (i.e., those who fulfills the constraints posed in RelBAC policy).

Nevertheless, we believe that encrypting users' resources is not enough in order to implement a cloud-based secure information sharing in a DSN. Indeed, as properly emphasized in [59], social network providers are able to perform data mining and user profiling analyzing the social graph as well. As such, so as to properly protect users' privacy, it is necessary to define strategies that hide the social graph from the social network provider. However, at the same time, these strategies have to leave to the social network provider the possibility to enforce RelBAC policies and such enforcement, trivially, requires to check the existence of connections in the social graph. Then, we believe it is necessary to define a decentralized anonymization technique for the social graph such that it both protects users' privacy and it allows the DSN provider to perform privacy-preserving analysis over it.

RelBAC policy recommender. In order to limit the RelBAC policies misconfiguration, we believe that DSN users should be assisted in the RelBAC policy definition process. Indeed, as highlighted in previous Section 1.1, misconfigured RelBAC policy may lead to harmful data release and may expose the privacy of others as well. Thus, we believe it is necessary to define a tool that guides DSN users in the definition of their RelBAC policies, offering customized suggestions tailored on each user's attitude towards his/hers own privacy.

Unfortunately, the definition of a RelBAC policy is influenced by many factors. Indeed, each user has a different attitude towards his/hers own privacy and this influences the RelBAC policies that he/she defines. Thus, to define a technology capable of helping users in their RelBAC policies definition, it is required to learn and model each user's privacy attitude (i.e., his/hers attitude towards his/hers own privacy), so as to exploit the learned preferences to define precise and customized RelBAC policies.

RelBAC flexible enforcement. As introduced before, misconfigured policies may lead to an improper data release as well as to deny someone the access over information that he/she should be, somehow, authorized. Whenever this occurs, then, the DSN provider is not able to fulfill requests coming from legitimate requestors.

Since a RelBAC policy recommender may not be enough to solve the policy misconfiguration problem, we believe that the DSN RelBAC enforcement engine should be augmented with a flexible and secure strategy capable of releasing information even when an explicit authorization is not granted. At first, it is necessary to extend the DSN RelBAC enforcement engine so as to recognize any possible *"implicit authorization"*, that is, the presence of information that can be harmlessly released in absence of an explicit authorization. Then, we plan to define a strategy to enhance the DSN RelBAC enforcement engine by exploiting these implicit authorization, making the engine capable of releasing harmlessly data for which no explicit authorization exists. However, our proposed approach does not plan to release any information for which there exist an explicit denial or any other kind of data whose release may lead to harmful situations.

For all the above discussed reasons, we based our researches on the improvement of the DSN reference architecture, as depicted in Figure 1.2, trying to overcome the above mentioned issues. In particular, working on a distributed and decentralized architecture, we set up our researches on the definition of tools and techniques that could be easily integrated with existing technologies.

1.3 Contributions

In this thesis we present our research contributions that we developed so as to enhance the DSN architecture. Our principal contributions are gathered in the followings.

1.3.1 Cloud-based secure information sharing.

As a first step we have been working on the definition of tools and protocols capable of integrating a cloud-based secure relationship-based information sharing model inside a DSN. As introduced in Section 1.2 we believe that a DSN should offer the following features, with which we had to deal with the definition of a cloud-based secure information sharing system:

Cloud-based encrypted data repository: DSN users' resources (e.g., UGCs, users' profiles, social graph) have to be stored as encrypted data into the public cloud, in that DSN users have to securely store their data.

Cloud-based anonymized social graph: The social graph has to be protected so as to prevent the social network provider to perform user profiling over such data. Then, it is necessary to define an anonymization technique such that it both protects the users' privacy and it allows the DSN provider to perform privacy-preserving analysis over it. As such, it is required to define a social graph anonymization strategy which:
(i) enables each DSN user to anonymize his/hers local view of the social graph

(i.e., his/hers direct contacts);
(ii) gives the possibility to combine privately all the anonymized views of the social graph into a deeper view of the anonymized social graph;
(iii) preserves the possibility to verify the existence of a certain connection between DSN users.

Privacy-preserving RelBAC enforcement: Besides protecting users' resources and the social graph, the RelBAC enforcement has to be performed so as to prevent any harmful information release. As such, the DSN provider cannot infer any information concerning the resource owner or the requestor user during the evaluation of a RelBAC policy.

In order to realize the above mentioned requirements, we extended the current DSN architecture by means of a 4-party architecture (see Figure 1.2). By means of these 4 entities, then, we realized the above introduced requirements as described in the followings.
At first, in order to let the DSN provider perform a privacy-preserving RelBAC enforcement, we designed a collaborative anonymization technique for the social graph. This technique allows to model as an algebraic polynomial the local view that every user has of the social graph, hiding a users's relationships in the polynomial's coefficients. The composition of this polynomial enable each DSN user to represent, in an anonymized form, all of his direct contacts and, at the same time, makes it easy to verify wether a connection between two users exists or not.
Combining together all the polynomials computed by the DSN users into a centralized party is then possible to reconstruct the social graph without any information leakage. The centralization of these data, which is performed into a dedicated entity stored into the public cloud, allows the DSN provider to perform a privacy-preserving RelBAC enforcement on the obtained data structure that represents an anonymized form of the social graph.
We then extended this collaborative anonymization technique with a distributed encryption protocol so as to define a secure system in which users' data (i.e., both relationship information, UGCs and users' resources) are securely transmitted and data can be exchanged without any information leakage. More details concerning the realization of the above introduced architecture along with the description of the communication protocols between the entities is gathered in Chapter 2.

1.3.2 Privacy settings recommender

After the formalization of the above described architecture, we have been researching over the definition of a technique capable of helping DSN users in defining RelBAC policies that fully reflects their own attitude towards privacy. As such, we have been working on a privacy settings recommender able to assist users in defining tailored and customized RelBAC policies. As briefly presented in Section 1.2, we planned to design a RelBAC policies recommendation system such that it can learn from DSN users their own attitude towards privacy, and exploits all the learned data to assist DSN users in the definition of RelBAC policies by suggesting customized privacy rules.

Our target, then, consisted in learning users' habits and preferences with respect to privacy management, so as to suggest customized access control policies. In achieving this goal, we had to keep into account that each individual performs a different decision process to decide how a resource has to be released in DSN. In addition to this subjective aspect, we also acknowledged that the decision an individual might take on resource release is greatly impacted by resource's contents. Based on these observations, we designed a recommendation system such that, given a certain DSN user, at first it learns the correlation that exists between a resource properties and the RelBAC policy that he/she is used to define for sharing resources with that property values. Then, each time a DSN user wants to upload a new resource into the platform, the recommender exploits the learned correlations to select the policies that are related to topics of the new resource. Finally, it generates a new access control policy by combining the retrieved ones, returning to the considered user as a customized suggestion. A detailed discussion over our proposed policy recommendation system is gathered in Chapter 3.

1.3.3 Enhance system utility through query rewriting

Although, beside exposing DSN users information, misconfigured RelBAC policies may also lead to situations in which certain data are not released to users which, under given circumstances, are able to discover them anyway. Whenever this occurs, then, the DSN provider is not able to fulfill requests coming from legitimate requestors. Then, in order to increase the DSN users utility, in terms of resources correctly released to legitimate requestors, we decided to cope with the RelBAC misconfiguration problem from a different point of view.

With more details, we first defined the concept of *"implicit authorization"* so as to denote all those information that could be safely released. As such, we defined as implicitly authorizable all those data for which do not exists an explicit access control policy authorizing their release but they can be harmlessly released nonetheless. Thus, this harmless release is realized by means of the existence of implications able to extend an existing access control policy from some explicitly authorized data to the implicitly authorized data. Then, we designed a technique capable of exploiting all the existing data dependencies (i.e., any correlation between elements) as a mean for increasing the system utility, that is, the number of queries that can be safely answered. As such, we defined a query rewriting technique capable of extending defined access control policy authorizations by exploiting data dependencies, in order to authorize unauthorized but inferable data.

1.4 Thesis Organization

Whereas this chapter summarizes the objectives and the contributions of our researches about the improvement of DSN architecture, additional details concerning the introduced techniques are provided in the following chapters. Details concerning our implementation of a cloud-based secure information sharing protocol for DSN are collected in Chapter 2. As such, Section 2.2 presents the reference decentralized architecture, Section 2.4 presents the introduced communication protocols whereas Section 2.3 includes a technical discussion about the collaborative anonymization technique. In Section 2.5 is provided a security analysis concerning the presented techniques, Section 2.6 presents *SocialCloudShare*, a sample implementation of the proposal presented in the Chapter, and Section 2.7 includes the results of the experiments provided for our proposal.
Chapter 3 provides a further extension of the tools presented in Chapter 2, introducing and detailing our contributions for what concerns the realization of a RelBAC recommendation system. With more details, Section 3.2 describes the learning techniques underlying the proposed recommender, which is presented in details in Section 3.3. At last, Section 3.4 collects the experimental results concerning the policy recommender.
Then, Chapter 4 includes the results of our researches that have been developed with the aim to increase DSN users utility, in terms of released information. As such, Chapter 4 describes query rewriting techniques designed to increase the amount of information correctly released by the DSN provider. Thus, Section 4.2 describes in detail the concept of implicit authorization and Section 4.3 presents the approach which prevents to release harmful information. The presented query rewriting technique is described in Section 4.4 whereas Section 4.5 with a technical discussion concerning its security, along with a discussion comparing the *Truman* and the *non-Truman models*, included in Section 4.6 Section 4.7 concludes the Chapter, collecting the results of the experiments with highlight the feasibility of the presented query rewriting technique.
At last, then, Chapter 5 revises the state of the art for all the topics treated in this thesis, whereas Chapter 6 concludes this thesis recapitulating our achievements and introducing our plans for future extensions of our work.

1.5 Related Publications

Davide Alberto Albertini and Barbara Carminati. (2014). Relationship-based information sharing in cloud-based decentralized social networks. In *Proceedings of* *the 4th ACM conference on Data and application security and privacy* (CODASPY '14), pages 297-304. March 3-5, 2014. San Antonio, TX, USA. ACM.

Davide Alberto Albertini, Barbara Carminati, and Elena Ferrari. (2014). Social-CloudShare: a Facebook Application for a Relationship-based Information Sharing in the Cloud. *EAI Endorsed Transactions on Collaborative Computing*, Volume 1, Issue 2. EAI.

Davide Alberto Albertini, Barbara Carminati, and Elena Ferrari. (2016). An extended access control mechanism exploiting data dependencies. *International Journal of Information Security* (IJS), pages 1-15. Springer.

Davide Alberto Albertini, Barbara Carminati, and Elena Ferrari. (2016). Privacy Settings Recommender for Online Social Network. *Workshop on Privacy in Collaborative & Social Computing* (PiCSoC '16) - Co-located with 2nd IEEE International Conference on Collaboration and Internet Computing (CIC 2016). November 1-3, 2016. Pittsburgh, PA, USA. IEEE.

2
Cloud-based Secure Information Sharing

As introduced in 1.3.1, our first investigation concerned the possibility to extend current DSN architecture, as depicted in Figure 1.2, so as to implement a cloud-based privacy preserving information sharing for DSN services.

As such, as emphasized in Section 1.1, this lead to deal with several challenges.
At first, a cloud-based DSN should follow the same principles regulating the information sharing in actual OSNs, such as the possibility for DSN users to specify their own RelBAC rules by means of simple and intuitive tools. Then, in order to protect users' resources and users' privacy, it is necessary to provide cryptographic protocols suitable for a cloud-based RelBAC enforcement. This implies that, along with these protocols, it is necessary to define a key management technique which, given a certain encrypted resource, ensures that the resource decryption key can be accessed only by those users that fulfill the constrains posed in the RelBAC policy assigned to the considered resource. Moreover, we have to ensure DSN users that no one except authorized users are able to decrypt their resources. Additionally, we found it necessary to define a novel privacy-preserving anonymization technique for the social graph that leaves the possibility to verify the existence of a connection between two given nodes without disclosing any users' information. Indeed, despite users' data can be encrypted in order to preserve users' privacy, this is not a feasible approach for what concerns the social graph. Moreover, the graph anonymization already present in literature are not feasible for a RelBAC information sharing, in that they modify the graph topology by either clustering nodes

or modifying the number of edges in the graph.

At such, in order to let the DSN provider perform a privacy-preserving RelBAC enforcement, we designed a collaborative anonymization technique for the social graph. This technique allows to store into a polynomial the local view that every user has of the social graph. The composition of this polynomial enable each DSN user to represent, in an anonymized form, all of his direct contacts and, at the same time, makes it easy to verify wether a connection between two users exists or not. These anonymized contact lists, then, can be combined together into an anonymized social graph that fully reflects the original social graph without any information leakage. Moreover, the anonymized social graph leaves the DSN provider the possibility to perform a privacy-preserving RelBAC enforcement.

Along with the above mentioned collaborative anonymization process, we extended the current DSN architecture by means of a 4-party architecture, so as to fulfill the above mentioned requirements.
As such, for what concerns the implementation of a secure and cloud-based encrypted data repository, we decided to *(i)* encipher any user resource directly at client-side by means of a Cipher Service (CS) and to *(ii)* split the process of generation of the encryption keys between two separate additional entities of the extended architecture. In particular, the key generation scheme implies that only CS is able to compute the key, whereas the two entities which generates the parameters from which the encryption key is derived are not able to compute it. Moreover these two entities, which are the Rule Manager Service (RMS) and the Key Manager Service (KMS), are in charge of, respectively, storing the users' RelBAC policies and storing into the cloud-based storage service the users' resources. At last, we introduced an entity named Path Finder Service (PFS) in charge of creating the anonymized social graph as well as inquiring it.

Furthermore, we run a set of experiments on Facebook OSN that gave encouraging feedbacks from what concerns user experience in a real life usage of our proposals.

2.1 Reference Model

Before presenting details of our proposal, we need to introduce some background knowledge that we will exploit throughout this thesis. Since the reference scenarios for our proposal are Decentralized Social Network platforms, in order to clarify the notation that we will use in this thesis, we summarize here the most relevant concepts of a DSN.

2.1.1 Social Graph model

In general, an Social Network platform is based on a Social Graph SG that models the users' relationships. In our model we exploit a directed labeled weighted graph $SG = (V, E, R)$ defined as follows:

V is a set of vertices, where each vertex $v_u \in V$ uniquely represents a certain DSN user u;

$E \subseteq (V \times V \times R \times (0, 1])$ is a set of edges, where each edge $e \in E$ models the existence of a relationship of type $r \in R$ leading from v_{from} to v_{to}, to which v_{from} assigned a trust value in $(0, 1]$;

R is a set of relationship type labels, denoting the nature of the relationships to which the label r is assigned.

Over a SG as defined above, we are interested in handling the paths connecting two non adjacent vertices. As such, we denote a path $p(x, y)$ which leads from v_x to v_y as a finite list of edges $(e_0, ..., e_n) \subseteq E^n$ such that the first edge e_0 starts from v_x, the last edge e_n leads to v_y, each couple of edges (e_i, e_{i+1}) shares one vertex (i.e., e_i leads to the vertex v which e_{i+1} starts from), and all the edges composing $p(x, y)$ have the same relationship type label r. With reference to Figure 2.1, as example, node A is connected to G with direct friendship and to L with a friendship path of length 3. For each path

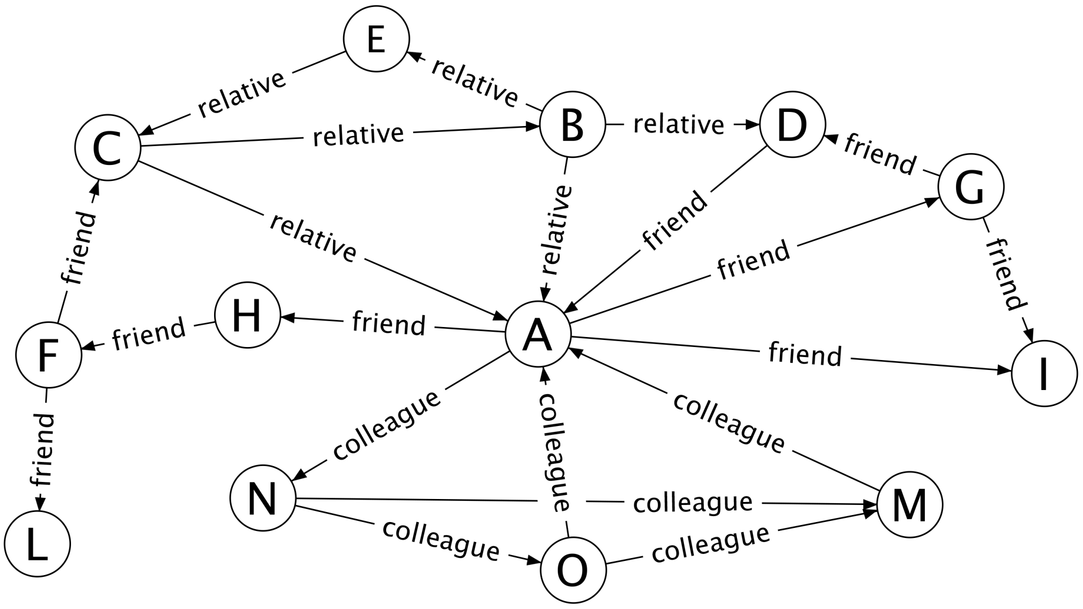


Figure 2.1: Example of labeled directed social graph

p existing in SG we can compute its length (i.e., the number of edges that compose

the path) and the trust value of the path, which can be calculated according to several algorithms present in literature (e.g., TidalTrust [37], EigenTrust [46], Advogato [47], *et al.*).
To keep the notation as light as possible, in the remainder of this thesis we will exploit a dot notation so as to denote the values of the 3 components of a given path $p(x, y)$, i.e. $p.head \equiv x$, $p.tail \equiv y$, and $p.trust$.

2.1.2 Access control model

In general, an access control policy specifies who is authorized to access, i.e., *Subject*, which is the authorized resource, i.e., *Object* and under which mode the access is authorized. In the following, we formalize an access control policy for OSN, according to the XACML standard [1] and by acknowledging that relationship based access control model (RelBAC) [20, 34] is the reference paradigm for controlled information sharing in Social Network platforms.

Definition 2.1.1 *An access control policy acp is defined as following:*

$$acp = \langle \texttt{Grantor}, \texttt{Subject}, \texttt{Object}, \texttt{Permission}, \texttt{Sign} \rangle$$

where:

`Grantor` *is the OSN user who defined acp;*

`Subject` *specifies users that satisfy acp. These can be represented as: 1) a list of OSN user identifiers, denoted as Sub_{ids}; 2) based on RelBAC model proposed in [21], a tuple Sub_{tuple}=$\langle RelType, MinTrust, MaxDepth \rangle$ that represents all those users that hold a relationship (i.e., an edge over SG) with* `Grantor` *of type specified by RelType, with a trust value greater or equal than $MinTrust$ and represented on SG by a path with length less or equal than $MaxDepth$;*

`Object` *is the identifier of resource protected by acp;*

`Permission` *specifies which actions (e.g., read, write, delete) are granted/denied over* `Object` *by means of acp;*

`Sign` *identifies whether acp is a positive or negative policy (e.g., $\oplus$ identifies a positive policy, $\ominus$ identifies a negative policy).*[1]

[1]To keep the notation as light as possible, throughout this thesis we will denote the components of a given access control policy *acp*, respectively, as *acp*.`Grt`, *acp*.`Sbj`, *acp*.`Obj`, *acp*.`Per`, and *acp*.`Sign`. In the remainder of this chapter, for simplicity, we will not take into account the `Permission` component

Notice that we introduce the authorization sign, that in RelBAC model [21] has not been considered, as all authorizations are implicitly positive. Indeed, we take this decision based on the fact that in many commercial social network services it is possible to define negative policies (e.g., "I authorize all of my friends but one"). In the remainder of this thesis we will assume deny-override paradigm, according to which a given requester v is granted to access a requested resource r iff there exists an access control policy acp which explicitly grants him/her and, at the same time, there exists no other access control policy acp' that explicitly forbids v to access the same resource.

Example 2.1.1 *Let Bob be a user subscribed to a given OSN. Let assume that Bob is used to share his pictures in the OSN with access control policies that in compliance with the followings:*

$$acp_1 = \langle Bob, \langle \text{“}friends\text{”}, 0.6, 1\rangle, \cdot, \texttt{READ}, \oplus\rangle \tag{2.1}$$
$$acp_2 = \langle Bob, \langle \text{“}colleagues\text{”}, 0.5, 1\rangle, \cdot, \texttt{READ}, \ominus\rangle \tag{2.2}$$
$$acp_3 = \langle Bob, \langle \text{“}relative\text{”}, 0.7, 1\rangle, \cdot, \texttt{READ}, \oplus\rangle \tag{2.3}$$

where the pictures to be uploaded are specified each time in the policies. As such acp_1 authorizes Bob's friend to access the uploaded pictures, acp_2 prevents that Bob's colleagues see the pictures, and acp_3 authorizes Bob's close relatives ($trust \geq 0.7 \wedge distance \leq 1$) to see Bob's pictures.
As well, a meaningful example of negative policies can be found in the followings:

$$acp_4 = \langle Bob, \langle \text{“}friends\text{”}, 0.6, 1\rangle, \cdot, \texttt{READ}, \oplus\rangle \tag{2.4}$$
$$acp_5 = \langle Bob, \langle \&Alice\rangle, \cdot, \texttt{READ}, \ominus\rangle \tag{2.5}$$

where the picture uploaded by Bob are granted by means of acp_1 to Bob's friends whose trust value is $\geq$ than 0.6, but in no case Alice (identified in the social network realm by hers identifier, here denoted as &Alice) is authorized to see Bob's pictures.

Given a resource rsc, its owner specifies a rule $\mathcal{R}$ according to which rsc has to be released in the OSN community. More precisely, a rule $\mathcal{R}$ is a finite set of access control policies defined according to Definition 2.1.1. For simplicity, hereafter, we assume that access rules consist of a unique access control policy acp, i.e., $\mathcal{R} = \{acp\}$.

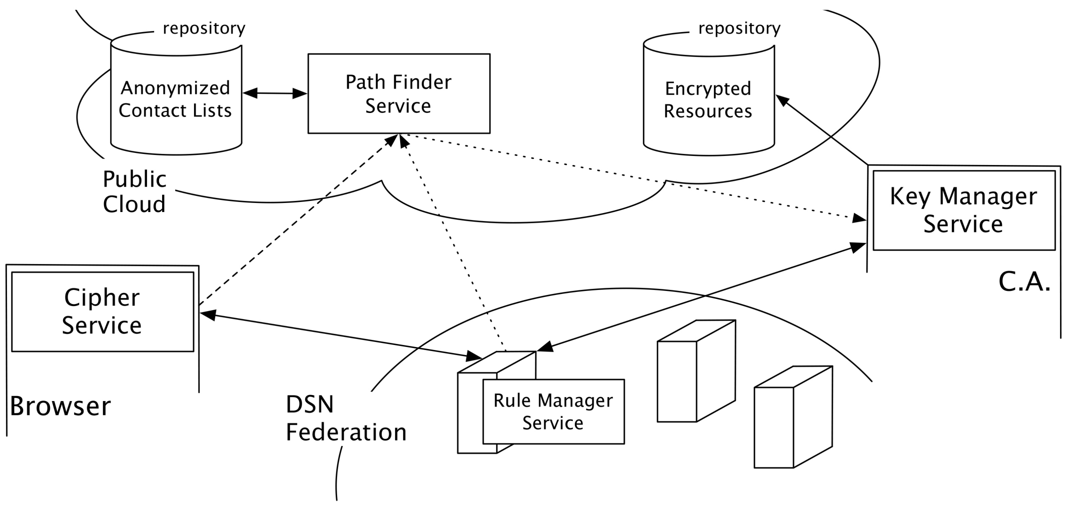


Figure 2.2: Cloud-based DSN reference architecture

2.2 Cloud-based DSN

According to the proposed architecture (see Figure 2.2), resources to be shared are locally encrypted by owners and stored into cloud storage (i.e., *Encrypted Resources* repository in Figure 2.2). In support of this, we assume that at the user is provided with the *Cipher Service* (*CS*) browser plugin, which is mainly in charge of owner's resources encryption and generation of anonymized owner's contact lists. Rule enforcement is carried out by releasing encryption keys only to requestors that satisfy at least one of the owner access rules. This enforcement requires the presence of two more entities: *Rule Manager Service* (*RMS*) to handle access control requests and *Key Manager Service* (*KMS*) for the keys management. As it will be discussed in Section 2.5, protocols regulating the resources release have been designed so that no one of *RMS* and *KMS* can decrypt owner resources as well as infer any information on owner relationships. This holds under the assumption that *RMS* and *KMS* do not collude together. In support of this assumption, we assume that *RMS* is implemented as a service in a DSN server; whereas *KMS* is an external trusted entity, whose role could be played by a Certificate Authority.

Moreover, to determine if a rule is satisfied, it is required to find those paths in the social network graph that connect owner to requestor. To protect relationship privacy, this path finding is carried out on anonymized data structures stored in the cloud (i.e., *Anonymized Contact Lists* (ACLs) in Figure 2.2). These data structures consist of an

of access control policies, assuming it to be unique and equal for any access control policy. However, the proposal presented in this chapter can be easily extended so as to consider and handle any possible value for this component.

entry for each user storing, in a private way (i.e., as polynomials), the list of users that are connected with him/her at different depths. As described in Section 2.3, these data structures are computed by a collaborative anonymization process where each user sends an anonymized version of its direct contact list (i.e., a polynomial) that is then privately combined with other anonymized contact list at cloud side so as to obtain ACLs. This task is performed by the *Path Finder Service* (PFS) at cloud side.

2.3 Collaborative Graph Anonymization

The goal is to design an anonymization strategy such that given a requestor r and a rule $\mathcal{R} = (t, d)$ it is possible to verify if it exists a path of type t and length less or equal than d connecting the resource owner to r. In order to support this query, we define a set of anonymized data structures that organize users' contacts by relationship type and length of the connecting paths. More precisely, given a relationship type t and a user u, the proposed anonymized contact list $ACL_t^d(u)$ privately encodes identifiers of those users that are connected with u by a path of t type and length equal to d. Before presenting its definition, let us introduce some needed notations.

We assume that with each user is assigned a set of identifiers based on the types of the relationships he/she involved. More precisely, given a user $u \in V$ in G, and a relationship type t, the unique type-based identifier for u is defined as $uid_u = h(id_u||t)$, where id_u is the unique identifier associated with u in the DSN realm.[2] Moreover, given a user $u \in V$ in G, we denote with $CL_t^n(u) \subset V$ the set of users that have with u a relationship of type t and depth n. Hereafter, we denote with $UID_t^n(u) = \bigcup_{v \in CL_t^n(u)} \{uid_v\}$ the set containing all the identifiers of these users.

Definition 2.3.1 *The* **Anonymized Contacts List** *for $CL_t^n(u)$, denoted as $ACL_t^n(u)$, is the set of coefficients of the polynomial $P_{CL_t^n(u)}$ of degree $|CL_t^n(u)|$, computed as follows:*

$$P_{CL_t^n(u)}(x) = \prod_{uid \in UID_t^n(u)} (x - uid). \tag{2.6}$$

As such, $uid \in UID_t^n(u)$ are all and the only possible roots of $P_{CL_t^n(u)}$, and $P_{CL_t^n(u)}$ results as a complete monic polynomial.

Example 2.3.1 *Based on social graph depicted in Figure 2.1, user A generates $CL^1_{friend}(A)$=$\{D, G, H, I\}$, $CL^1_{relative}(A)$ =$\{B, C\}$ and $CL^1_{colleague}(A)$=$\{M, N, O\}$. According to Definition 2.3.1, the ACLs for user A are defined as the coefficients of the*

[2]This relies on the assumption that inside DSN each user is uniquely identified by an unique id, which is true in each commercial Social Network.

following polynomials:

$$\begin{aligned}
P_{CL^1_{friend}(A)}(x) =&(x - h(id_D||friend)) \cdot (x - h(id_G||friend)) \cdot \\
&\cdot (x - h(id_H||friend)) \cdot (x - h(id_I||friend)) \\
P_{CL^1_{relative}(A)}(x) =&(x - h(id_B||relative)) \cdot (x - h(id_C||relative)) \\
P_{CL^1_{colleague}(A)}(x) =&(x - h(id_M||colleague)) \cdot (x - h(id_N||colleague)) \cdot (x - h(id_O||colleague))
\end{aligned}$$

Based on Definition 2.3.1, it is easy to verify if a node v satisfies an access rule $ac = (t, d)$ defined by user u. Indeed, it is required only to evaluate whether v's identifier, i.e., $h(id_v||t)$, is a root of at least a polynomial $P_{CL^n_t(u)}(x)$, with $n \leq d$. In contrast, under the assumption of a high degree of P it is hard to find all uids (see Section 2.5).

As Example 2.3.1 highlights, ACL^1_t can be easily computed at user side. Unfortunately this does not hold for ACL^n_t with $n > 1$, since it requires to know $uids$ of each indirectly connected users, whereas in this proposal we assume that users have only a local view (i.e., they only know their direct neighbors). To overcome this limitation, we propose a solution where ACLs for $n > 1$ are privately generated directly at cloud side by PFS.

The basic idea is that, given a certain $ACL^1_t(u)$ generated by u on his own, if v is a direct contact of type t for u,[3] then, for the same relationship type t, v's contacts at distance d have to be considered $(d{+}1)$-hops contacts of u and vice versa, for any d¿1. Thus, if $u \in CL^1_t(v)$, $\forall d$¿1, then $CL^d_t(u) \subseteq CL^{d+1}_t(v)$. As consequence, $ACL^{d+1}_t(v)$ has to contain $uids$ encoded in $ACL^d_t(u)$ and $ACL^{d+1}_t(u)$ has to contain those which are encoded in $ACL^d_t(v)$.

To privately insert $ACL^d_t(v)$'s roots into $ACL^{d+1}_t(u)$, we exploit the polynomial property that, given $P(x)$ and $P'(x)$, the roots of the polynomial $Q(x){=}P(x) \cdot P'(x)$ are all and only values in the union set of roots satisfying $P(x)$ and roots satisfying $P'(x)$.

As such, we assume that each time a anonymized contact list $ACL^1_t(u)$ for a new user u is received, PFS first verifies which of the already registered users, i.e., their uids, satisfy the new uploaded polynomial. That is, it determines the subset $S \subset V$ such that $\forall p \in V,\ P_{CL^1_t(u)}(uid_p) = 0 \iff p \in S$. Then, $\forall p \in S$: (1) it inserts the p's direct contacts into anonymized 2-hop contact list of u, that is, it stores into $ACL^2_t(u)$ the coefficients of polynomial resulting by the multiplication of $P_{CL^2_t(u)}(x) \cdot P_{CL^1_t(p)}(x)$; (2) it inserts the p's indirect contacts of distance n into anonymized (n+1)-hop contact list of u, i.e., $ACL^{n+1}_t(u)$, as coefficients of polynomial resulting by the multiplication of $P_{CL^{n+1}_t(u)}(x) \cdot P_{CL^n_t(p)}(x)$.

It is important to note that, step (2) could be performed for any n value. However, we limit this computation at threshold $MaxDepth$, which indeed constraints the maximum

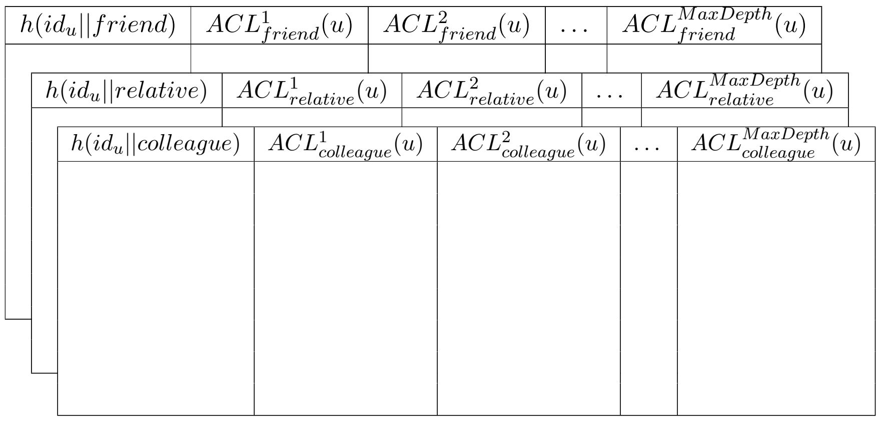
[3]That is, $uid_v = h(id_v||t)$ satisfies $P_{CL^1_t(u)}(x = uid_v) = 0$.

distance between two nodes in the anomyzed graph. As such, this value limits the length of the paths over which searches have to be performed. We believe this does not represent a limitation. Indeed, according to the small world property and other studies (e.g., [25]) we expect that every pair of users (u,v) is connected by a limited number of edges in the SN community.

Example 2.3.2 *Let us consider the generation of ACL encoding a indirect friends list at 2-hops distance. Given $P_{CL^1_{fr}(A)}(x)$ of Example 2.3.1, the polynomial $P_{CL^2_{fr}(A)}(x)$ is generated as follows:*

$$\begin{aligned}
P_{CL^2_{friend}(A)}(x) =& P_{CL^1_{friend}(D)}(x) \cdot P_{CL^1_{friend}(G)}(x) \cdot P_{CL^1_{friend}(H)}(x) \cdot P_{CL^1_{friend}(I)}(x) = \\
=&[(x - uid_A) \cdot (x - uid_G)] \cdot \\
&\cdot[(x - uid_A) \cdot (x - uid_D) \cdot (x - uid_I)] \cdot \\
&\cdot[(x - uid_A) \cdot (x - uid_F)] \cdot \\
&\cdot[(x - uid_A) \cdot (x - uid_G)] = \\
=& (x - uid_A)^4 \cdot (x - uid_D) \cdot (x - uid_F) \cdot (x - uid_G)^2 \cdot (x - uid_I).
\end{aligned}$$

We assume that ACLs are stored into a set of tables, one for each supported relationship type. More precisely, for each relationship type $t \in RT$, the corresponding $Table_t$ contains an entry for each user u storing into $MaxDepth$ columns the corresponding $ACL^n_t(u)$, with $1 \leq n \leq MaxDepth$.



$h(id_u \| \| friend)$	$ACL^1_{friend}(u)$	$ACL^2_{friend}(u)$	...	$ACL^{MaxDepth}_{friend}(u)$

$h(id_u \| \| relative)$	$ACL^1_{relative}(u)$	$ACL^2_{relative}(u)$	...	$ACL^{MaxDepth}_{relative}(u)$

$h(id_u \| \| colleague)$	$ACL^1_{colleague}(u)$	$ACL^2_{colleague}(u)$	...	$ACL^{MaxDepth}_{colleague}(u)$

Figure 2.3: ACL Tables structure

2.4 Relationship-based Information Sharing

Let us now introduce how the proposed framework enforces the relationship-based information sharing by illustrating protocols for resource upload and download. In doing that, we assume that communication between entities is transmitted over secure channels.[4] We assume that the public key of CS, K^{+}_{CS}, is known by RMS and KMS as it will be used to encipher messages directed to CS.

Resource Upload Given a resource rsc, before uploading it in the cloud, its owner, say user u, has to encrypt it using a symmetric key K_{rsc}. This key is computed as combination of two secrets, i.e. $K_{rsc}= \mathcal{F}(secret_u,\ secret_{rsc})$, which are separately generated by RMS and KMS.[5] These secrets are defined such that: $secret_u$ is uniquely associated with user u by RMS; $secret_{rsc}$ is uniquely associated with rsc by KMS.

Thus, before any upload, resource owner has to interact with RMS and KMS so as to retrieve the corresponding $secret_u$ and $secret_{rsc}$. The protocol underling this message exchange is represented in Table 2.1. Note that, as depicted also in Figure 2.2, CS communicates with KMS by means of RMS. Indeed, we adapted the structure of Needham-Schroeder protocol. As such, CS creates a message for RMS, by encapsulating inside it the message directed to KMS. Assuming that only KMS has the private key K^{-}_{KMS}, RMS cannot decrypt the encapsulated message, but just forward to KMS.

Thus, once the *secrets* are generated by RMS and KMS (messages 2, 4 in Table 2.1) and received by u (message 6 in Table 2.1, encrypted with CS public key), the user is able to compute K_{rsc}. Hence, u composes a message including the encrypted resource (to be transmitted to KMS) and the set of access control rules $\mathcal{R}_{rsc}$ that RMS has to store. As depicted, CS sends all messages to RMS, which then forwards nested encrypted messages to KMS. After the execution of protocol depicted in Table 2.1, the cloud data storage service contains the encrypted resource, whereas RMS and KMS contain only resource metadata. In particular, KMS stores id_{rsc} and $secret_{rsc}$, whereas RMS saves id_{rsc} along with the resource access control rules $\mathcal{R}_{rsc}$.

Resource Download If a requestor req wishes to download and decrypt rsc, it sends a message to RMS with the corresponding ids (i.e., message 1 in Table 2.2). This retrieves the corresponding access rules $\mathcal{R}_{rsc}$, the id of rsc's owner (i.e., id_{own}). Then, assuming $\mathcal{R}_{rsc} = ac = (t, d)$, it inquires PFS to search for a path connecting the requestor to the owner, with all edges labeled with t and length less than d (i.e., messages 3-4 in Table 2.2). It is important to note that if PFS sends the yes/not answer back directly to RMS, this might bring to some information leakage. Indeed, for some particular access rules, knowing whether this is satisfied gives exact information on existing paths.

[4]If available and certificated, the CS uses the user's private-public key to encrypt messages; otherwise when a new CS instance is created, the KMS generates and certifies a couple of asymmetric keys $\langle K^{+}_{CS}, K^{-}_{CS}\rangle$ which are then communicated to the CS.

[5]Several $\mathcal{F}$ functions can be adopted. In actual implementation, we make use of `XOR`.

1.	$CS \rightarrow RMS$:	$\{id_u, N_1, \{id_u, id_{rsc}, N_2\}_{K^+_{KMS}}\}_{K^+_{RMS}}$
2.	RMS:	Retrieve $secret_u$, if any, or generate a new one
3.	$RMS \rightarrow KMS$:	$\{id_u, id_{rsc}, N_2\}_{K^+_{KMS}}$
4.	KMS:	Compute new unique $secret_{rsc}$
5.	$KMS \rightarrow RMS$:	$\{secret_{rsc}, N_2 + 1\}_{K^+_{CS}}$
6.	$RMS \rightarrow CS$:	$\{secret_u, N_1 + 1, \{secret_{rsc}, N_2 + 1\}_{K^+_{CS}}\}_{K^+_{CS}}$
7.	CS:	Compute $K_{rsc} = \mathcal{F}(secret_u, secret_{rsc})$
8.	$CS \rightarrow RMS$:	$\{id_u, N_1 + 2, id_{rsc}, \mathcal{R}_{rsc}, \{id_u, id_{rsc}, \{rsc\}_{K_{rsc}}, N_2 + 2\}_{K^+_{KMS}}\}_{K^+_{RMS}}$
9.	RMS:	Store $\mathcal{R}_{rsc}$
10.	$RMS \rightarrow KMS$:	$\{id_u, id_{rsc}, \{rsc\}_{K_{rsc}}, N_2 + 2\}_{K^+_{KMS}}$
11.	KMS:	Store $rsc_{K_{rsc}}$ in cloud storage service

Table 2.1: Resource upload protocol

As example, let assume that A in Figure 2.1 has set up the rule $ac = (friend, 1)$ for a given resource. If D requests this resource, RMS will receive a positive answer from PFS, which makes RMS able to understand that between D and A there exists a direct friendship. In order to avoid this leakage PFS answers are sent only to KMS (i.e., message 5 in Table 2.2). Based on the answer, then KMS generates a *token* containing the $secret_{rsc}$ if the answer is a positive, i.e., there exists a path satisfying the rule, or a random number for negative answer. Together with the `URL` from which resource can be downloaded, the token is then encrypted with CS public key and send it to RMS (i.e., message 6 in Table 2.2).

The `URL` sent from KMS to CS is a temporarily valid `URL` provided by cloud storage service at KMS requests. The rsc to be downloaded is reachable at this `URL` only for a small and fixed lapse of time, afterwards rsc is moved back to the private realm of the storage service, without any public access.[6] RMS adds $secret_{own}$ into the received message, encrypts with CS public key, and forwards it to CS (i.e., message 7 in Table 2.2). Then, user decrypts $secret_{own}$ and $token$ values, and generates $\mathcal{F}(secret_{own}, token)$, which returns the correct encryption key K_{rsc} only if the $token = secret_{rsc}$, that is, only if KMS receives positive answer from PFS, confirming the existence of a path satisfying the rule.

[6]Moving resources on temporary `URL` is a common approach used by several cloud storage services (e.g., Dropbox) to limit access at requested resources.

1.	$CS \rightarrow RMS$:	$\{id_{req}, id_{rsc}, N_1\}_{K^+_{RMS}}$
2.	RMS:	Retrieve ac_{rsc} and id of rsc's owner, i.e., id_{own}
3.	$RMS \rightarrow PFS$:	$\{h(id_{own}\|\|ac_{rsc}.type), h(id_{req}\|\|ac_{rsc}.type), ar_{rsc}, \{id_{req}, id_{rsc}\}_{K^+_{KMS}}\}_{K^+_{PFS}}$
4.	PFS:	Check if path (own, req) fulfills ac_{rsc}
5.	$PFS \rightarrow KMS$:	$\{found, \{id_{req}, id_{rsc}\}_{K^+_{KMS}}\}_{K^+_{KMS}}$
6.	$KMS \rightarrow RMS$:	$\{token, URL_{rsc}\}_{K_{CS^+}}$ where $token{=}secret_{rsc}$ if $found \equiv$ `true`, $token{=}random$ otherwise;
7.	$RMS \rightarrow CS$:	$\{secret_{own}, N_1 + 1, \{token, URL_{rsc}\}_{K^+_{CS}}\}_{K^+_{CS}}$
8.	CS:	Compute K_{rsc} with received *secrets*, then download and decrypt resource, if possible.

Table 2.2: Resource download protocol

2.5 Security Analysis

Except of KMS, which is assumed to be trusted, the framework is evaluated in a honest-but-curios adversary model, as such CS, PFS, and RMS parties are assumed to properly follow the protocols, by trying at the same time to infer sensitive information.

2.5.1 Relationship-based Resource Sharing

Given a resource rsc owned by own, and the corresponding set of access rules $\mathcal{R}_{rsc}$, the framework has to ensure that a requestor $req \in V$ can access (i.e., decrypt) rsc if and only if at least an access condition $ac \in \mathcal{R}_{rsc}$ is satisfied, that is, if there exists at least a path connecting req and own whose relationship type and depth satisfy at least an access condition $ac_{rsc} \in \mathcal{R}_{rsc}$.

According the proposed architecture a user can access a resource rsc if it has the corresponding decryption key K_{rsc}. Let us show that secrets are distributed only to authorized users, i.e., users satisfying at least an access condition. Let assume that user req does not satisfy any of the access conditions in $\mathcal{R}_{rsc}$, but after the execution of protocol in Table 2.2, it received the correct secrets to compute K_{rsc}. Since req does not satisfy any rule and assuming that the protocol is fairly executed, at the end of execution req receives $secret_{own}$ and $token{=}random$ (see messages 6,7 in Table 2.2). This does not allow him to compute the encryption key $K_{rsc} \neq \mathcal{F}(secret_{own}, random)$. Under the assumption that $\mathcal{F}(\cdot, \cdot)$ is collusion-free $\mathcal{F}(secret_{own}, random) \neq \mathcal{F}(secret_{own}, secret_{rsc})$, as such a contradiction arises.

Note that the above statement considers only malicious internal users of DSN. For other attackers, the only way to access resources is to decrypt it. At this purpose an attacker has to obtain both the secrets. From Table 2.1, we can see that these *secrets* are locally generated by RMS and KMS, and securely transmitted to CS in message 8 (or, similarly, in message 7 of Table 2.2). As such, to achieve $secret_{own}$, the message has to be decrypted with private key of requestor, whereas to access $secret_{rsc}$, two decryptions have to be performed. Under a honest-but-curios model we need to consider also the involved parties (i.e., PFS, and RMS, as KMS is trusted). However, the considerations made above for external attackers still hold, with the difference that RMS already knows $secret_{own}$. Moreover, it is not able to decrypt $\{secret_{rsc}, N_2 + 1\}_{K_{CS}^+}$ in message 5 of Table 2.1, hence it cannot retrieve $secret_{rsc}$ and build K_{rsc}.

Further relevant analysis is related to key reuse and resources update. Let suppose that a granted user *req*, following the protocol of Table 2.2, is authorized to download and decrypt *rsc*. Even if *req* knows the value of K_{rsc}, he cannot obtain the new `URL` value that is sent in message 6 of Table 2.2. Moreover, in case the resource or the corresponding access rule is changed, the framework considers the modified resource rsc' as a new resource, with the consequent generation of new $secret_{rsc'}$.

2.5.2 Relationship Information Leakage

According to the proposed framework, relationship information is stored at cloud side in the form of ACLs. As such, under the assumption that all messages are securely exchanged, we see only two ways to infer information on existing relationships: (1) inferring roots, i.e., *uids*, directly from ACLs, and (2) inferring information from the access request evaluation. Let see in more details both these two attacks.

Inference from ACLs

Tracing ACLs insertions. A first attack can be performed by PFS service. This service receives directly from users the ACLs for their direct contacts, ACL^1. Then to built/update ACL^n with n≥2, PFS tries all possible *uids* for finding users with mutual friends, so as to propagate them in other ACLs. According to an honest-but-curious behavior, PFS could trace every computation result, obtaining thus roots/uids for each uploaded ACL^1. Having these uids, PFS could rebuild a set of graphs $\mathcal{G}=\bigcup_{t\in RT}\{G_t\}$. All these graphs are, topologically, subgraphs of the original graph G, except for their nodes' ids that are different from the one in G. Thus, under the assumption of a collusion-free $h(\cdot)$ function, PFS is not able to merge together the graphs in $\mathcal{G}$, since the same user appears in these graphs with different uids.

As firstly discussed in [5], a graph where only ids are anonymized (i.e., randomized) cannot provide enough privacy protection, as it can suffer of re-identification attacks. This kind of attack is possible under the assumption that the attacker knows the existence of given structures H, such as Hamiltonian graphs, in the actual graph. As such, since the graph topology is preserved, if the attacker finds H in a graph G_t, he/she can use this match to re-identify all nodes connected with that structure and, in turn, their contacts of relationship type t and so on. With respect to this attack, thus, we have to make two considerations. The first is that is hard for an attacker to identify these structures when the original structures can be split in more substructures. A further consideration is that, in order to perform the attack, that is, in order to search of a given structure in subgraphs G_t, if possible, the PFS has to know which structure it has to look for. This would require to access to some information of the real graph G, as example, by registering in the social platform (as in the attack discussed in [5]) or by colluding with some existing social network users. However, this is not possible under the assumption that PFS is curious but honest.

Direct *uids* computation from ACLs. This kind of attacks can be run: by an external entity that maliciously retrieved $Table_t$ from the cloud storage; by the cloud manager that gets malicious access to $Table_t$, under the assumption of untrusted public cloud. Given ACLs tables, the attackers can try the following two approaches: *solving polynomial* or extracting their roots by *brute force*. For both aspects, thus, it's important to emphasize a common aspect of these approaches, that is the ground field over which the polynomials are built.

Solving polynomials. According to Definition 2.3.1, polynomials in $Table_t$ are generated such that they are in a complete monic form. This allows us to refer to the *Abel-Ruffini Theorem* that states that for equations with degree greater or equal than 5 there is no algebraic formula for root-finding. Note that we can reasonably assume that all our polynomials have a degree greater than 5, that is, all ACLs encode more than 5 contacts.[7]
Despite this, there exist both numerical and arithmetical ways for the polynomial root-finding problem [56]. But almost all of them are inefficient with high degree polynomials. For example, given a polynomial P of degree n, *Vincent-Akritas-Strzeboński* (*VAS*) method requires to compute at least n variable substitutions and transformations in a recursive approach. Furthermore, it is relevant to observe that polynomials in Definition 2.3.1 are similar to Wilkinson's polynomials [69], in that they are complete monic polynomial. As such, it also holds the Wilkinson property stating that root finding problem

[7]For those polynomials with a lower degree, we can assume that CS adds some fake roots.

is generally ill-conditioned. Finally, since our polynomials are in the form

$$P(x) = \prod_{i=0}^{n}(x - a_i) \Rightarrow P(x) = \sum_{i=0}^{n} b_i x^{n-i},\ b_0 = 1,\ b_n \neq 0$$

the last coefficient b_n is defined as the multiplication of all the roots. As such, an alternative approach for root-finding requires to factorize b_n. Still, both root finding problem and integer factorization problem are well known to be hard problems. To the best of our knowledge, the most efficient ways to retrieve polynomial roots are the deterministic algorithms or randomized algorithms presented in [36], which have polynomial-time complexity. The required time, thus, may become really significant once the ground field over the problem are defined increases in space.

However, if an attacker exploits one of these techniques, the time and computational resources needed to retrieve the real graph make this kind of attack infeasible. Indeed, as previously discussed, retrieving roots is only the first step, that is, it only allows the attacker to find the set of anonymized graphs $\mathcal{G}$. Then the re-identification attack in [5] has to be performed. As such, we believe the time needed to carry out the overall re-identification attack is greater than the possible graph lifespan and utility time.

Brute force. To retrieve *uid*s from an ACL, attackers can have a brute force approach trying all possible *uid*s so as to verify which of them satisfy the ACL polynomial. A first thing to emphasize is that the time estimation for this attack highly depends on the dimension of the social network. More precisely, let assume that all edges have the same relationship type, as such all relationships are encoded in an unique table with an entry for each user in G. This brings to a scenario where the ACL table contains $|V| \cdot MaxDepth$ polynomials. In order to reconstruct the social graph it is enough to find the solutions for polynomials that encode users' direct contats, that is, $|V|$ polynomials. To retrieve direct contacts of a target user u, that is, to find the roots of polynomial $P_{CL^1(u)(x)}$ it is required, in the worst-case, $(|V| - 1) \cdot t_{eval}$, where V is set of users in G and t_{eval} is the time of polynomial evaluation. In fact, to solve a single polynomial out of $|V|$, the attacker has to evaluate such polynomial with $|V| - 1$ possible *uid*s . To reconstruct the whole graph, this process has to be repeated for every other user in V, without repetition. This brings to a time estimation of

$$\sum_{i=1}^{|V|-1} \left[(|V| - i) \cdot t_{eval}\right] = t_{eval} \sum_{i=1}^{|V|-1} (|V| - i) =$$
$$= t_{eval} \sum_{i=1}^{|V|-1} (i) = \frac{|V|(|V| - 1)}{2} t_{eval} \sim O(|V|^2) t_{eval}.$$

As example, in a DSN with $|V| = 1 \cdot 10^6$ and with $t_{eval} = 10ms$ (see Table 2.3), the attack requires $\sim 3.17 \cdot 10^2$years. As such, we believe this attack is not feasible.

A final note holding for solving polynomials attack as well as for the brute force is about the ground field over which the polynomials are built. Indeed, $h(\cdot)$ has to be selected such to have a codomain large enough to contain, at least, $|V| \cdot |RT|$ different values. It is expected that $h(\cdot)$ generates up to 10^{12}, as example, which are then used for *uid*s. Even though most of calculators may compute values up to $log_2(10^{12}) < 64$ bit, it is not true that they are able to perform a polynomial evaluation, e.g., during a brute force attack, since these operations imply several exponentiations.

Inference from access rule enforcement.

For some particular access rules, knowing if this is satisfied gives exact information on existing paths. To avoid this possible leakage, the resource download protocol has been designed such that neither KMS or RMS know both the query path and the answer. However, inference from access rule evaluation could be exploited by external attacker. As example, with reference to Figure 2.4, let's assume that an attacker wishes to know whether between u and v there exists a relationship of type t. The attacker has to:

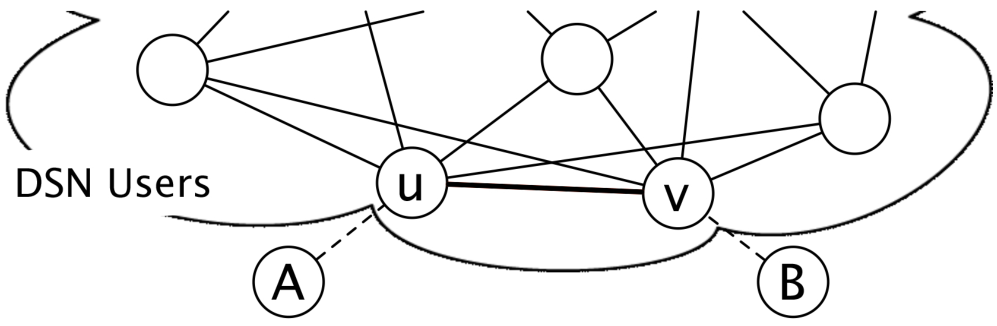


Figure 2.4: Inference Scenario Scheme

(1) create two fake social network users A and B; (2) A invites u to establish a new relationship, and B to v; (3) if both the relationships have been established, according to protocol in Table 2.1, A uploads a certain resource rsc with the following access rule $ac = (t, 3)$; (4) according to protocol in Table 2.2, B requests rsc: if the access is granted the attacker will know that between u and v it exists a direct relationship of type t.

Unfortunately this attack is possible as this information gain is intrinsic in the problem we are addressing (i.e., an information release regulated by existence of given relationships). However, there are some important considerations: (1) in order to accomplish the attack, both relationship requests have to be confirmed. This is less likely (i.e., half percentage) than to those attacks, like [5, 48] that assume to insert a subgraph H in G by connecting only a fake user with a real user; (2) with no clue on u and v, this attacks becomes really expensive. Indeed the attacker needs to successfully establish $2 \cdot |E| \cdot |RT|$ new relationships to discover the whole graph topology. Moreover, in a real scenario, a social network user confirms only a small percentage of relationship requests he/she receives from unknown people, and a refused relationship request cannot be replayed endlessly. Thus, a malicious entity needs an overwhelming effort to perform his attack.

2.6 SocialCloudShare

In order to evaluate the effectiveness of the hereby introduced techniques, we developed *SocialCloudShare (SCS)*. SCS has beed designed as a Facebook application able to act, according to the above described architecture, as a Rule Manager Service.

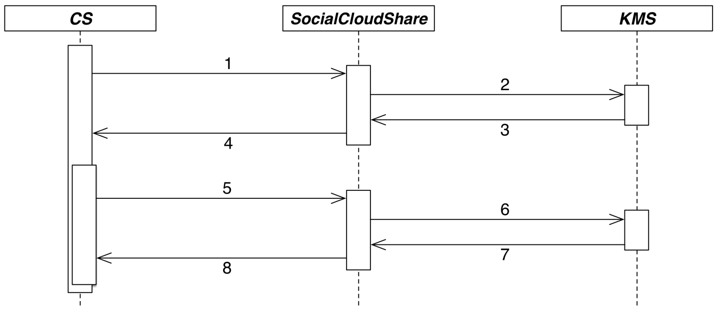
Since both the RMS and the others architecture entities were designed to act in a decentralized architecture, we had to modify some details while implementing them. Nevertheless, though, we tried to keep each entity as compliant as possible to its decentralized declaration. We decided to implement SCS as a Facebook application instead of making a proper decentralized example architecture so as to collect the more data as possible concerning a real life usage of the platform. Moreover, thanks to the Facebook API system, such implementation required much less efforts than implementing a complete decentralized example. All the data collected by our example, then, are gathered in Section 2.7.

2.6.1 Communication Protocols

As a first step, we will introduce how the communication protocols between the architecture entities have been implemented, illustrating the messages exchanged in each step. Although this is not declared explicitly, we assume that each communication between entities is transmitted over secure channels. Beyond encryption primitives present in messages schemas, we assume that HTTPS connections can be instantiated before communicating, so that an additional security layer can be granted.
To keep the notation as light as possible, we will denote with K a symmetric encryption key, with K^+ and K^- a public and private key, and with $K^{session}$ a session key valid only for the current communication session. For any key, we report as subscript the framework component for which the key has been generated (e.g., K^+_{SCS} denotes a public key generated for *SocialCloudShare*). Moreover, we denote with K_{rsc} a resource encryption key, whereas $secret_{owner}$ and $secret_{rsc}$ denote the secret tokens that are used for the generation of the resource encryption keys. Finally, with such defined keys, we denote with $\{message\}_K$ a message that is encrypted exploiting K as encryption key.

Login Phase

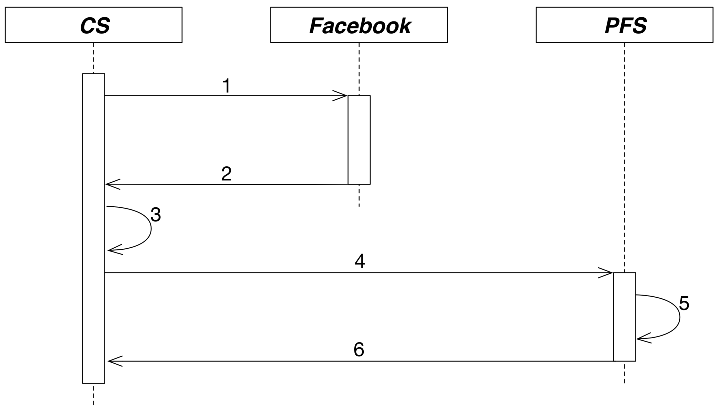
Since we assumed that no CS instance is aware of *SocialCloudShare* and KMS public keys when firstly initialized, the communication is initalized by requesting K^+_{SCS}, K^+_{KM}, where K^{+}_{SCS} and K^{+}_{KM} respectively denote the public keys of *SocialCloudShare* and the *Key Manager* (see messages 1-4 in Figure 2.5).



1. $\{id_{user}$, "PUBLIC_KEYS_REQ"$\}$
2. $\{id_{user}$, "PUBLIC_KEYS_REQ"$\}$
3. $\{id_{user}, K^{+}_{KM}\}$
4. $\{id_{user}, K^{+}_{KM}, K^{+}_{SCS}\}$
5. $\{id_{user}, K^{session}_{SCS}, \{id_{user}, K^{session}_{KM}\}_{K^{+}_{KM}}\}_{K^{+}_{SCS}}$
6. $\{id_{user}, K^{session}_{KM}\}_{K^{+}_{KM}}$
7. $\{id_{user}$, "SESSION_KEY_RECEIVED"$\}_{K^{-}_{KM}}$
8. $\{id_{user}$, "SESSION_KEY_RECEIVED", $\{id_{user}$, "SESSION_KEY_RECEIVED"$\}_{K^{-}_{KM}}\}_{K^{-}_{SCS}}$

Figure 2.5: Login phase: messages exchange

Once the user has received them, the CS generates a pair of 128 bit random keys, denoted as $K^{session}_{SCS}$ and $K^{session}_{KM}$, that will be exploited as session keys for the user current session. Note that, as depicted by the architecture in Figure 2.2, CS communicates with KMS relying only on *SocialCloudShare* (which is acting as RMS), since there exist no direct communication channel between the CS and the KMS. Indeed, we adapted the structure of Needham-Schroeder protocol (see [55]). As such, when the CS has to communicate with the KMS, it creates a message for *SocialCloudShare* and encapsulates inside this the message directed to KMS. *SocialCloudShare*, when receives such message, forwards to KMS the encapsulated chunk (e.g., messages 5,6 in Figure 2.5). Assuming that only KMS has the private key K^{-}_{KM}, *SocialCloudShare* cannot decrypt the encapsulated message, but it has just to forward it to the KMS.[8] Once both *SocialCloudShare* and KMS correctly receive the session key, they reply to the user with messages 7-8 in Figure 2.5.



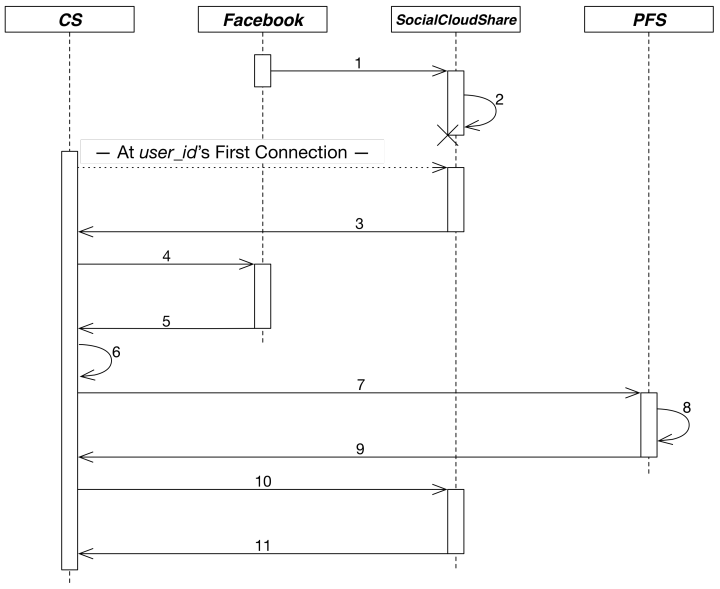
1. FB.api{ 'me/friends', {fields: 'id'} }
2. JSON-formatted CL^1
3. ACL^1 := anonymize(CL^1)
4. $\{id_{user}, ACL^1\}_{K^+_{PFS}}$
5. storeAndPropagate(id_{user}, ACL^1)
6. { "ACK" }$_{K^-_{PFS}}$

Figure 2.6: Registration phase: messages Exchange

User Registration

Figure 2.6 depicts the message exchange when users execute the application for the first time. Exploiting Facebook JavaScript SDK, the user's contact list is requested (message 1) and gathered directly from the social network provider (message 2) with no need to rely on any intermediate service. The anonymization process (message 3 in Figure 2.6) produces ACL^1, taking as input the direct contact list CL^1, at user side; as such, no relationship data are sent to the provider before being anonymized. The anonymized contact list is then sent to the PFS, which stores it and propagates in all the ACLs (see message 5 in Figure 2.6).

The message exchange is ended with a response message produced by the PFS, i.e. message 6, to notify the CS that the protocol has been properly executed by both parties and the sent data have been successfully handled.



1. {'id': 'user_id', 'changed_fields': 'friends' }
2. storeUpdate(user_id, "CL_OUT_OF_DATE")
3. { "CL_OUT_OF_DATE" }
4. FB.api{ 'me/friends', {fields: 'id'} }
5. JSON-formatted CL^1
6. ACL^{1-new} := anonymize(CL^{1-new})
   $ACL^{1-removed}$:= anonymize($CL^{1-removed}$)
7. $\{id_{user}, ACL^{1-new}, ACL^{1-removed}\}_{K^+_{PFS}}$
8. updateAndPropagate($id_{user}, ACL^{1-new}, ACL^{1-removed}$)
9. { "ACK" $\}_{K^-_{PFS}}$
10. { "CL_UPDATE_DONE" $\}_{K^{session}_{SCS}}$
11. { "ACK" $\}_{K^{session}_{SCS}}$

Figure 2.7: Contact list update: messages exchange

Contact List Update

Figure 2.7 summarizes the messages exchange when the users' contact lists are modified (i.e., by adding or removing relationships). In the current implementation, we exploit Facebook *Real Time Updates (RTU)*.[9] RTU is a feature of Facebook Graph

[8]This still relies on the assumption that *SocialCloudShare* and KMS do not collude.

[9]https://developers.facebook.com/docs/graph-api/real-time-updates/v2.0 .

API[10] which allows Facebook thirdy-party Social Apps to be informed, directly from the OSN provider, when certain pieces of data change (e.g., new profile pictures, new friendship requests). With this functionality, *SocialCloudShare* does not need to continuously keep syncronized with the social graph, because a callback function is called, by means of an HTTP POST request, every time a user changes his/her own contact list (see message 1 in Figure 2.7).

Unfortunately, the OSN only notifies *SocialCloudShare* about the changed fields, without revealing any other information. As such, it is then necessary to fetch from the OSN social graph all the data about new or removed friends. For this reason, we designed *SocialCloudShare* to keep track of all those users whose contact lists are not syncronized with the ACLs stored at Cloud side. Then, when each of those users make use of *SocialCloudShare*, he/she receives a message that informs the CS that the contact list has to be syncronized (see messages 2,3 in Figure 2.7). Exploiting JavaScript functions, the current contact list is fetched from the social graph and new users (or, equivalently, removed users) are detected. CL^{1-new} and $CL^{1-removed}$ denote the two contact lists computed by the CS representing the lists of the new contacts and the removed contacts. By exploiting the *anonymize* function in message 6 of Figure 2.6, CL^{1-new} and $CL^{1-removed}$ are anonymized.

Then, the user sends to the PFS these two separate ACLs (or just one of them, in case the other one results in an empty list) (see message 7 in Figure 2.7). The PFS runs again the process of ACL propagation, adding the new information whenever these data are missing, or removing old information in case of relationship removal (i.e., by dividing polynomials instead of multiplying them). The messages flow is concluded with a special flag (see messages 9-11), in order to inform both the PFS and *SocialCloudShare* that the protocol has been properly executed.

Resource Upload

The messages exchange for the resource upload phase follows the same schema as the one depicted in Figure 2.5, whereas the messages content is depicted in Figure 2.8.

Before uploading a certain resource rsc in Dropbox, its owner, say user u, has to encrypt it using a symmetric key, that is, K_{rsc}. This key is computed as combination of two secrets, i.e. $K_{rsc}= \mathcal{F}(secret_{owner}, secret_{rsc})$, which are separately generated by *SocialCloudShare* and the KMS.[11] In the given implementation, we choose to encrypt resources exploiting `AES-256` algorithm [54], operating in Cipher Block Chaining (CBC) mode [30], where the plaintext is padded according to PKCS#7 [45]; for this reason we designed the two secrets with length of `256` bit. As it will be discussed later, these secrets are released to a requestor by *SocialCloudShare* and KMS if and only if he/she

[10]https://developers.facebook.com/docs/graph-api/ .

[11]Several $\mathcal{F}$ functions can be adopted. In our implementation, we make use of `XOR`.

1. $\{id_{user}, id_{rsc}, \text{“RSC\_UPLOAD\_REQ”}, \ldots$
   $\ldots\{id_{user}, id_{rsc}, \text{“RSC\_UPLOAD\_REQ”}\}_{K_{KM}^{session}}\}_{K_{SCS}^{session}}$
2. $\{id_{user}, id_{rsc}, \text{“RSC\_UPLOAD\_REQ”}\}_{K_{KM}^{session}}$
3. $\{id_{user}, id_{rsc}, secret_{rsc}\}_{K_{KM}^{session}}$
4. $\{id_{user}, secret_{user}, \{id_{user}, id_{rsc}, secret_{rsc}\}_{K_{KM}^{session}}\}_{K_{SCS}^{session}}$
5. $\{id_{user}, id_{rsc}, \mathcal{R}_{rsc}, \{id_{user}, id_{rsc}, \{rsc\}_{K_{rsc}}\}_{K_{KM}^{session}}\}_{K_{SCS}^{session}}$
6. $\{id_{user}, id_{rsc}, \{rsc\}_{K_{rsc}}\}_{K_{KM}^{session}}$
7. $\{id_{user}, id_{rsc}, \text{“RSC\_STORED”}\}_{K_{KM}^{session}}$
8. $\{id_{user}, id_{rsc}, \text{“ACR\_STORED”}, \{id_{user}, id_{rsc}\text{“RSC\_STORED”}\}_{K_{KM}^{session}}\}_{K_{SCS}^{session}}$

Figure 2.8: Resource upload phase: messages exchange

satisfies at least one access rule condition associated with *rsc*.

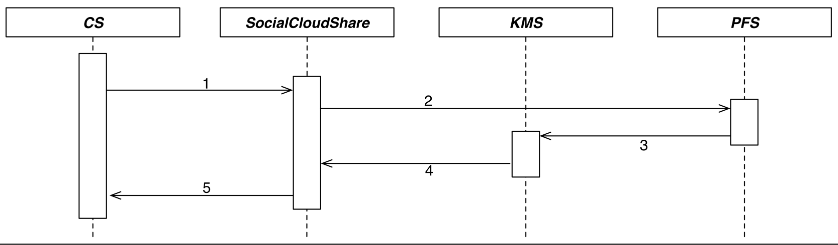
Thus, before any upload, resource owner has to interact with both *SocialCloudShare* and the *KMS* so as to retrieve the corresponding $secret_{owner}$ and $secret_{rsc}$. Assuming the user shares a symmetric session key only with *KMS*, negotiated during the login phase, *SocialCloudShare* cannot decrypt the encapsulated message and thus cannot discover $secret_{rsc}$. Once the secrets have been generated by *SocialCloudShare* and the *KMS*, they are received by u encrypted with pre-shared session key (see message 4 in Figure 2.8); as such, the user is able to compute K_{rsc}. Hence, u composes a message including the encrypted resource (to be transmitted to *KMS*) and the set of access control rules $\mathcal{R}_{rsc}$ that *SocialCloudShare* has to store. In our implementation $\mathcal{R}_{rsc}$ is a 1 byte value; the 5 more significant bits translate the relationship type (with a maximum of possible relationship types equal to 32) and the 3 less significant bits translate the maximum depth value of the access control condition. Even though our implementation currently supports only *“friend”* relationships, this implementative choice leaves the framework ready to further improvements.

As depicted in Figure 2.8, the *CS* sends all messages to *SocialCloudShare*, which then forwards nested encrypted messages to the *KMS*. After the execution of the protocol illustrated in Figure 2.8, the Cloud data storage service contains the encrypted resource, whereas *SocialCloudShare* and the *KMS* contain only resource metadata. In particular, the *KMS* stores id_{rsc} and $secret_{rsc}$, whereas *SocialCloudShare* saves id_{rsc} along with the resource access control rules $\mathcal{R}_{rsc}$, where id_{rsc} denotes a unique identifier for the resource.

Resource Download

In order to enforce a relationship-based resource sharing, the framework has to release encryption keys only to requestors satisfying at least an access control rule associated with the requested resources. To determine if an access rule is satisfied, the PFS service is inquired. To protect the communication between $SocialCloudShare$, the KMS, and the PFS we assume there exists a symmetric encryption key, denoted as K_{PFS}, shared between those three entities. By using this key, the communication encrypted with K_{PFS} cannot be decrypted by anyone unless the components of the framework.

If a requestor req wishes to download and decrypt rsc, it has to send a message to $SocialCloudShare$ with the related ids (message 1 in Figure 2.9). $SocialCloudShare$ retrieves the corresponding access rules $\mathcal{R}_{rsc}$ and the id of rsc's owner (i.e., id_{own}). Then, assuming for simplicity $\mathcal{R}_{rsc}$ contains only one access control condition $acc = (t, d)$, it inquires the PFS to search for a path connecting the requestor to the owner, with all edges labeled with t and length less than d (i.e., message 2 in Figure 2.9). It is important to note that if the PFS sends the yes/no answer back directly to $SocialCloudShare$, this might bring to some information leakage. Indeed, for some particular access rules, knowing whether the rule is satisfied gives exact information on existing paths. As such, the answer produced by the PFS is sent to the KMS (see message 3 in Figure 2.9).



1. $\{id_{req}, id_{own}, id_{rsc},$ "RSC_DWNLD_REQ"$, \ldots$
   $\ldots \{id_{own}, id_{rsc},$ "RSC_DWNLD_REQ"$\}_{K_{KM}^{session}}\}_{K_{SCS}^{session}}$
2. $\{h(id_{req}||acc_{rsc}.type), h(id_{own}||acc_{rsc}.type), acc_{rsc}.depth \ldots$
   $\ldots \{id_{own}, id_{rsc},$ "RSC_DWNLD_REQ"$\}_{K_{KM}^{session}}\}_{K_{PFS}}$
3. $\{result, \{id_{own}, id_{rsc},$ "RSC_DWNLD_REQ"$\}_{K_{KM}^{session}}\}_{K_{PFS}}$
4. $\{token_{secret}, \texttt{URL}_{rsc}\}_{K_{KM}^{session}}$
5. $\{id_{req}, secret_{own}, \{token_{secret}, \texttt{URL}_{rsc}\}_{K_{KM}^{session}}\}_{K_{SCS}^{session}}$

Figure 2.9: Resource download phase: messages exchange

The `URL` sent from the KMS (see message 4 in Figure 2.9) is a temporarily valid `URL` provided by the Cloud storage service upon KMS requests. The rsc to be downloaded is reachable at this `URL` only for a small and fixed interval of time, afterwards rsc is moved back to the private realm of the storage service, without any public access.[12] Message 4 (see Figure 2.9) contains, along with the above mentioned `URL`, the value of $token_{secret}$, which is $token_{secret} = secret_{rsc}$ in case the PFS sent a positive answer, or a random value otherwise. $SocialCloudShare$ inserts $secret_{own}$ into the received message, encrypts it with pre-shared session key and forwards it to the user (i.e., message 5 in Figure 2.9). Then, the user decrypts $secret_{own}$ and $token_{secret}$ values and generates $\mathcal{F}(secret_{own}, token_{secret})$, which returns the correct encryption key K_{rsc} only if $token_{secret} = secret_{rsc}$, that is, only if the KMS receives a positive answer from the PFS, confirming the existence of a path satisfying the rule.

2.6.2 Cipher Service - Browser Plugin

The *Cipher Service (CS)* is the component in charge of client-side resource encryption and of anonymized contact lists generation. We choose to implement the abovedmentioned functionalities with a set of JavaScript functions, in order to achieve a better usability than a customized software and to give users the possibility to make use of it with no restriction given by his/her operative system.

By exploiting jQuery library[13] and AJAX-like[14] techniques, CS is able to process user actions (e.g., mouse clicks, page requests, upload/download requests) inside $SocialCloudShare$. The most important functionalities offered by CS are the encryption primitives for resources/messages encryption/decryption. For what concerns the resource encryption/decryption phase, the CS can be seen as a cipher black box. Thus, plaintext resource is taken as input and coded into a ciphertext resource and vice versa. As such, no entity except the CS takes part in these processes. At this purpose, we decided to exploit an existing library, named Crypto-JS,[15] available under BSD-3 License on Google Code, offering several encryption primitives ready to be used. In particular, for resources encryption, we exploit AES-256 algorithm applied according to Cipher Block Chaining (CBC) mode, where the plaintext is padded according to PKCS#7.

In order to exploit CBC mode, an Initialization Vector iv is necessary during the encryption and decryption phases. For this reason, the CS generates each time a random value as initalization vector (by exploiting *CryptoJS.lib.WordArray.random(128/8)*), which is added prior to the ciphertext, such that the iv itself can be securely stored along with the encrypted resource.

[12] Moving resources on temporary `URL` is a common approach used by several Cloud storage services (e.g., Dropbox, in this implementation) to limit access of requested resources.

[13] http://jquery.com/ .

[14] http://www.w3schools.com/ajax/default.ASP .

[15] https://code.google.com/p/crypto-js/ .

Figures 2.10 and 2.11 depict the functions used in the *CS* implementation.

```
CryptoJS.AES.encrypt(
  'Resource-Stream',
  'Resource-Secret-Key',
  { iv: 'iv',
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.Pkcs7
  }
);
```

Figure 2.10: Javascript AES encipher

```
CryptoJS.AES.decrypt(
  'Encrypted-Resource-Stream',
  'Resource-Secret-Key',
  { iv: 'iv',
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.Pkcs7
  }
);
```

Figure 2.11: Javascript AES decipher

Another important feature handled by *CS* is the generation of ACL^1. To compute such ACL^1, the JavaScript library contains functions implementing the polynomial multiplication, i.e., computing the discrete convolution bewteen number sequences. As first step, the direct contacts list is fetched from Facebook social graph, by means of Facebook JavaScript SDK. As depicted in Figure 2.12, the JavaScript SDK needs to be initalized with a valid *Social-App-Id*, which is the identifier assigned by Facebook when registering a Social App in the OSN realm.

The *CS* can request to Facebook, by means of the Javascript `FB` Object, the logged user's friend list (e.g., see messages 1,2 in Figure 2.6) so that it can receive the current user's direct contacts identifiers. By having these identifiers, the *CS* can generate the user's ACL^1. Once this ACL^1 is fully computed, it is sent to the *Path Finder Service*, which is the component in charge of handling the anonymized social graph.

```
<script type='text/javascript'>
  $(document).ready(function() {
    $.ajaxSetup({ cache: true });
    $.getScript('//connect.facebook.net/en_UK/all.js',
      function(){ FB.init({
        appId: 'Social-App-Id',
      });
    });
  });
</script>
```

Figure 2.12: Facebook JS-SDK Load Phase

2.6.3 Path Finder Service Implementation

As outlined above, the *Path Finder Service (PFS)* is the component of *SocialCloudShare* that handles the anonymized social graph. All the ACLs are stored into the *ACL Repository* table, where the record is in the form $[id_u, ACL^1(u), ACL^2(u), \ldots, ACL^{MaxDepth}(u)]$, that is, it contains the user identifier and all his/her ACLs of different path length (e.g., see Figure 2.3).

The PFS is implemented as a web service, by means of a Java servlet that handles HTTP requests. The request received from *EM* instances are encrypted with the PFS public key, i.e., K^+_{PFS}. On the other hand, requests received from *SocialCloudShare* entity are encrypted with a pre-shared session key, denoted as K_{PFS}, that grants a lower overhead than an asymmetric-key encryption.

Such component, like *SocialCloudShare* and KMS entities presented in the following sections, has been developed inside the Spring framework[16] and exploiting STS[17], an eclipse-based IDE.[18]

Algorithm 1 describes the procedure executed each time a new ACL^1 is received from a *SocialCloudShare* user. This algorithm makes use of the *ACL Repository*, denoted with R, and of a boolean matrix, *updates*, that keeps track of the ACLs that have been modified during the propagation procedure.

Each time a new ACL^1 is received, along with the user *id*, the PFS stores inside the

[16]http://projects.spring.io/spring-framework/ .

[17]http://spring.io/tools .

[18]Spring is an application framework with built-in modules that facilitate Java application development, in which code dependecies are directly handled by Apache Maven (http://maven.apache.org/) and Gradle (http://www.gradle.org/) at build-time, generating a `.jar` archive that can run under, for example, an Apache Tomcat (http://tomcat.apache.org/) web server.

ACL Repository those new information (see Line 3 in Algorithm 1) and sets as true the corresponding cell of the *updates* matrix (see Line 4). Once the data have been stored, the procedure analyzes, from the shallowest level to the deepest, the *ACL Repository* record (see Lines 5,7). We denote with $e.id$ the user identifier stored in the repository entry e, and with $e.ACL^d$, the ACL^d stored in the same reporitory entry.

For each record e, the procedure performs a second iteration over all different record e' (see Line 8). If the *updates* matrix contains `true` in the cell corresponding to e', the procedure performs a polynomial evaluation, where the polynomial is the ACL taken from e' and the user identifier is taken from e (see Line 9). In case the polynomial evalutation results 0, and each polynomial evaluation for smallest path length (see Lines 10, 11) result in a value different from 0, the information carried by $e.ACL^1$ and $e'.ACL^1$ is cross-propagated to level $d+1$, where d is the variable iterated over the path depth values (see Lines 12, 13). Along with the cross-propagation, the procedure updates the values of the *updates* matrix, that is, it keeps track of the above modified entries. Finally, a boolean variable *stop*, initally set with `true` (see Line 6), is set with `false` (see Line 16).

The above described procedure terminates when, given a path depth d, ACLs are no more modified throughout the whole iteration over the repository record, that is, the boolean value of the variable *stop* is `true` when the loop cycle at Line 7 ends, and the procedure is forced to terminate (see Line 18).

2.6.4 SocialCloudShare Implementation

Differently from *PFS* and *KMS*, *SocialCloudShare* has been developed with both a back-end system and a graphical interface, which is displayed when users access *SocialCloudShare* application inside Facebook.
SocialCloudShare back-end is implemented as a web application, designed according to the Model-View-Controller architectural pattern, such that a precise HTTP request on a given `URL` calls a certain method of the underlying servlet. The most relevant methods offered by *SocialCloudShare* are called by handling HTTP requests incoming on the following URLs:

SCS/ : A request to the base `URL` of the web application generates and returns *SocialCloudShare* homepage. The underlying controller, when necessary, fetches and stores some of the users' data (e.g., full name, profile picture). These data are collected interacting directly with the OSN provider, exploiting Facebook Graph API in order to receive users' profile information.

SCS/key/broadcast : This URL is requested automatically when the *CS* detects that the public keys of *SocialCloudShare* and the *KMS* are not stored at client-side.

```
Input: $id_u,\ ACL^1(u),\ ACL\ Repository\ R$
begin
    boolean[][] updates;
    R.push({ $id_u, ACL^1(u), 1, 1, \ldots$ });
    updates[1][u] = true;
    foreach $d \in \{1, 2, \ldots, MaxDepth\}$ do
        boolean stop = true;
        foreach entry $e \in R$ do
            foreach entry $e' > e$ do
                if $(updates[d][e'.id])$ AND $(e'.ACL^d(x = e.id) == 0)$ then
                    foreach $d' < d$ do
                        if $e'.ACL^{d'}(x = e.id) \neq 0$ then
                            $e.ACL^{d+1} = e.ACL^{d+1} \cdot e'.ACL^1$;
                            $e'.ACL^{d+1} = e'.ACL^{d+1} \cdot e.ACL^1$;
                            updates[d+1][e.id]=true;
                            updates[d+1][e'id]=true;
                            stop = false;
                        end
                    end
                end
            end
        end
        if stop then
            exit;
        end
    end
end
```

Algorithm 1: Anonymized Contact List Propagation Procedure

It represents the arrival point of message 1 in Figure 2.5. The controller forwards the received parameters to the KMS on its URL *KM/key/broadcast*.

SCS/key/negotiate : This URL is requested automatically when the CS detects that the session keys for communicating with *SocialCloudShare* and KMS are not stored at client-side. It represents the arrival point of message 5 in Figure 2.5. The controller forwards the message that is encapsulated in the received one, that is the message from CS to the KMS, on the URL *KM/key/negotiate*.

SCS/fb_updates : This URL is reachable both with HTTP GET and HTTP POST requests, but it is supposed to be requested only from Facebook provider. The application listens to information from the OSN, waiting for *Real Time Updates (RTU)*. Once an HTTP GET request has been received, the application communicates with Facebook in order to control and regulate the subscription for RTU. HTTP POST requests, on the other hand, are assumed to include information about the user activity in the OSN (e.g., a new profile picture, a new friendship in the social graph). The controller underlying these requests keeps track of those users that have a contact list that is not syncronyzed with the ACLs stored in the PFS (e.g., see message 1 in Figure 2.7).

SCS/upload : The controller that handles HTTP requests to this `URL` is the one responsible of starting the resource upload procedure. The received message is decrypted with the corresponding session key, and the encapsulated message (that cannot be decrypted by *SocialCloudShare*) is forwarded to the KMS on its URL *KM/upload*. Once the message is forwarded, the controller holds and waits for a response from the KMS.

SCS/upload/finalize : A request done to this URL finalizes an upload procedure already started. As such, the underlying controller waits for the answer and, once received, the message is decrypted and the encapsulated part is forwarded to the KMS. The remainder of the message, thus, includes the access control rule $\mathcal{R}_{rsc}$ of the uploaded resource. As such, $\mathcal{R}_{rsc}$ is stored by *SocialCloudShare* along with the resource owner identifier and the resource identifier.

SCS/download : The controller that handles the incoming requests to this `URL` is the one in charge of listening to download requests, representing the initialization of a download process. As such, this message gathers information about the *requestor*, the *owner*, and the *resource* involved in the download process. These data are sent to the PFS that, after checking the existence of a path on ACLs, sends the corresponding result to the KMS on its URL *KM/download*.

2.6.5 Key Manager Service Implementation

Similar to *SocialCloudShare*, the KMS has been developed as a web application, exploiting the STS IDE. On the other hand, KMS is sligthly different from the previously presented *SocialCloudShare*. The KMS is designed to listen to communication exclusively coming from *SocialCloudShare*, PFS, and Dropbox; as such the application performs an IP filtering prior to accept incoming data. In case some data are received by a peer that is not regognized as belonging to one of those three parties, its requests are rejected and the communication channel closed. Then, the KMS is designed without any front-end interface; any incoming message that is identified as valid brings the KMS to perform some data processing and the output is directly sent as response to the message sender. The main methods offered by KMS are called by handling HTTP requests incoming on the following URLs:

KM/key/broadcast : This URL can only be requested by *SocialCloudShare* (e.g., via IP filtering) and it is requested only when CS detects that the public keys of *SocialCloudShare* and the KMS are not stored at client-side. It represents the arrival point of message 2 in Figure 2.5.

KM/key/negotiate : This URL is requested automatically when the CS detects that the session keys for communicating with *SocialCloudShare* and KMS are not present at client-side. It represents the arrival point of message 5 in Figure 2.5.

KM/upload : The procedures that are run when this controller is called are the ones responsible of generating and storing the resource secret $secret_{rsc}$, where rsc represents the identifier of the resource that is going to be uploaded .

KM/upload/finalize : The underlying controller includes the methods for resources upload to the Cloud Storage Service (e.g., Dropbox). Once a resource is stored into Dropbox, the KMS locally stores certain resource metadata, such as the name, the filetype, and the last modification date. Those data are then displayed to users, through the *SocialCloudShare* GUI.

KM/download : This URL is listening only to requests coming from PFS. Indeed, the underlying controller is listening to messages resulting from a path search on the anonymized graph. Messages received at this `URL` not only contain the result of the path search performed by PFS, but they include pieces of data that were included in the dowload request sent from the requestor. With these information, the KMS is able to ask Dropbox to generate $\texttt{URL}_{rsc}$, that is a temporary valid `URL` for the encrypted resource download. $\texttt{URL}_{rsc}$ is included in the response along with $token_{secret}$, that may be a randomly generated value, in case the path finding returns a negative answer, or the value of $secret_{rsc}$ otherwise, where rsc is the downloaded resource.

2.7 Experimental Evaluations

To evaluate the framework performance, we carried out several tests. At this purpose, we set up Java classes to run the experiment exploiting *java.Math.BigInteger* package for instantiating polynomials coefficients to overcome the limitation due to blown of values dimension; polynomials and users data are then stored in a MySQL database. The workstation used is an Intel Core 2 Quad Q6600 @ 2.40 GHz $\times$ 4, with 8GB RAM. Each experiment ran exploiting Stanford SNAP datasets for Facebook.[19]

Our first experiment estimates the time required for the joining a new user to the anonymized graph. We recall that this implies that the new user u submits the anonymized list of its direct contacts, which is then elaborated by PFS to: (1) verify which of already registered users p are direct contacts of u; (2) propagate contacts of u in p's contact lists, and vice versa. As such, a parameter that greatly impacts this performance is the number of users that have been already registered in the graph. For this reason, the SNAP dataset has been split into two chunks of data: a $X\%$ of nodes for which the graph joining has to be performed, whereas the remaining are considered as already registered users. Experiments are carried out with 10%, 20% and 30% of the nodes to be inserted, which is a collection of about 1200 and 800 adjacency lists, selected out of the dataset randomly. As reported in Figure 2.13, for each experiments users are organized according their coverage in the graph (i.e., x-axis). More precisely, the experiment is carried out as follows. (1) First, a new user u is randomly selected. (2) Then, it is computed the number of nodes that are connected with u with a path of $MaxDepth$ distance,[20] we refer to this value as $RelTree(u)_{MaxDepth}$. Based on this value, it is computed the coverage of u's relationships in SNAP graph G_{SNAP}, as $c = RelTree(u)_{MaxDepth}/|G_{SNAP}.V|$.[21] We limit the $MaxDepth$ value to 5.[22] (3) We then insert u in the anonymzed graph, tracking the time t required to complete the insertion procedure. The resulting couples $(coverage, t)$ are collected in Figure 2.13.

From the result we can infer that the time required to insert u, in its worst-case is:

$$t_{insert}(u) \sim MaxDepth \cdot \left(\frac{|N| \cdot (|N|+1)}{2} \cdot t_{eval} \cdot 2t_{mult}\right).$$

Moreover, assuming the time necessary to perform a polynomial evaluation (t_{eval}) and a multiplication between polynomials (t_{mult}) approximated with constant values, this time is $\sim O(|N|^2)$, where N is set of users that have been already successfully inserted before u.

[19]`http://snap.stanford.edu/data/index.html#socnets`

[20]The nodes are counted without repetition, since a certain node may be repeated at different depths

[21]Where $|G_{SNAP}.V|$ is the set of nodes in SNAP, which is about 4000.

[22]This value has been selected also based on SNAP Dataset statistics that point the value 4.7 as the *90-percentile effective longest shortest path*.

To give more details on PFS performance, Table 2.3 presents the average time required for computing polynomial evaluations and multiplications during the user insertion. The values are divided, row by row, per depth d. These values may highlight how a polynomial evaluation requires generally short time, whereas a multiplication between polynomials requires more resources.

Level/depth	Polynomial Evaluation [ms]	Polynomial Multiplication [s]
1	10.777	20.072
2	498.921	110.972
3	1340.344	240.653
4	3207.567	305.899
5	6185.483	535.360

Table 2.3: Average time required for PFS operations

This result is encouraging, since we expect the system to perform much more polynomial evaluations than multiplications, in that we expect that a resource download should happen much more time than a user subscription in a legitimate scenario. Moreover, since an access control rule evaluation requires just a polynomial evaluation, Table 2.3 (i.e., the second column) represents also the average time for rule enforcement. This is further confirmed by the final experiment represented in Figure 2.14. We estimate the time required to evaluate several, randomly generated, access control rules, by varying the condition on maximum depth. As expected, the time increases with higher depth value, as this requires to find more path, that is, to perform more polynomial multiplications. The *polynomial evaluation* column in Table 2.3 represents also the average time required for an access control rule evaluation, since this requires a single polynomial evaluation.
At a final note, since users' relationships are grouped by their relationship types, the computational load of PFS due to multiplication is expected to be limited. Moreover this approach perfectly suits the cloud-based paradigm, because PFS can be replicated with small effort. Thus, a load-balancer entity may support PFS entities, and redirect requests to a non-locked service.

On the other hand, for what concerns the data gathered through *SocialCloudShare*, we have been able to depict the overhead introduced by our architecture on the social network usage experience. As such, in the following we gather information concerning messages encryption size, messages encryption average time, and resource encryption average time.

Message Encryption Size

Figure 2.15 depicts the overhead, in terms of length of messages, implied by messages encryption. The considered messages are those exchanged during the login, upload, and download phases (see Figures 2.5, 2.8, 2.9).

Each bar in Figure 2.15 represents a single message, in term of message size. Each message is denoted by a *message plaintext size*, that is, the size of the message before encryption and a *message overhead*, that is, the size of the message once encrypted. Some message includes an encapsulated message (see message 5 in Figure 2.5, messages 1,2 in Figure 2.8, and message 1 in Figure 2.9) that requires a further encryption phase prior to message encryption. For those messages, Figure 2.15 reports the *encapsulated message plaintext size* as well as the *encapsulated message overhead.* As such, the *message plaintext size* for those messages is composed of the message plaintext size, the encapsulated message plaintext size, and the encapsulated message overhead.

Finally, it is important to note that messages in the login phase (see Figure 2.5) are encrypted using an asymmetric key encryption scheme.[23] This motivates the higher overhead introduced by messages encryption in such phase. Messages exchanged during upload and download phases, on the other hand, are encrypted exploiting `AES-128` algorithm.

Message Encryption/Decryption Average Time

Tables 2.4 and 2.5 report the time consumption given by messages encryption. This experiment has been carried out monitoring the time required by encryption primitives to encrypt/decrypt the corresponding messages; for each message the encryption/decryption phase has been repeated 10 times. As such Tables 2.4, 2.5 report the minimum and the maximum time obtained in this experiment, along with the time average and the standard deviation.

With the exception of the 5^{th} message of the resource upload phase, the encryption/decryption primitives completed the execution in less than 40 milliseconds. In this experiment, we used a 64byte text file as uploaded resource to keep the simulation as light as possible. The average time for all messages encryption/decryption, thus, never reached a value higher than 30 milliseconds; as such this result let us state that the protocols may run with no impact on the user experience over the OSN.

[23]In particular, we exploit `RSA-1024` for this phase .

Protocol	Msg	Time [ms]			Standard
		Min	Avg	Max	Deviation
Login	5	3.0	6.0	9.0	1.89
	7	1.0	2.2	3.0	0.6
	8	2.0	3.0	6.0	1.61
Upload	1	6.0	14.4	24.0	7.68
	3	3.0	8.1	12.0	3.01
	4	6.0	15.0	24.0	8.16
	5	14.0	28.0	56.0	16.57
	7	2.0	5.2	8.0	2.4
	8	4.0	8.0	16.0	4.0
Download	1	6.0	16.2	24.0	7.61
	2	5.0	8.0	20.0	4.58
	3	3.0	8.1	12.0	3.01
	4	6.0	13.2	24.0	7.96
	5	9.0	26.1	36.0	10.22

Table 2.4: Time required for message encryption

Protocol	Msg	Time [ms]			Standard
		Min	Avg	Max	Deviation
Login	6	4.0	4.8	6.0	0.98
Upload	2	4.0	10.0	16.0	4.1
	6	8.0	21.6	32.0	8.8
Download	2	4.0	9.2	16.0	4.02
	3	5.0	11.0	20.0	6.63
	4	5.0	10.8	20.0	6.38

Table 2.5: Time required for message decryption

Resource Encryption Average Time

Finally, Table 2.6 reports the results of the experiment to estimate the encryption time necessary to prepare a resource to be uploaded. Unlike messages, which are encrypted exploiting `AES-128` algorithm, resources are encrypted exploiting `AES-256` algorithm, in order to achieve a better security for resources, that have to be stored in the public Cloud. The first column in Table 2.6 reports the size (in Mbytes) of the resource to be encrypted, while the other columns gather the time interval, in seconds, necessary to perform the encryption. In this experiment, we used random-generated `ASCII` strings, with pre-determined lengths. The encryption phase has been repeated 10 times for each

File Size	Time [s]			Standard
[Mb]	Min	Avg	Max	Deviation
0.2	1.92	2.01	2.08	0.05
0.4	3.66	3.94	4.14	0.15
0.6	5.76	5.94	6.16	0.13
0.8	7.2	8.07	8.72	0.56
1.0	9.71	9.99	10.3	0.18
1.2	11.47	12.03	12.58	0.41
1.4	13.8	14.05	14.28	0.16
1.6	14.77	16.05	17.36	0.85
1.8	17.16	17.83	18.57	0.46
2.0	19.54	20.01	20.59	0.33
2.5	23.53	25.34	26.45	0.96
3.0	28.3	30.33	32.34	1.25
3.5	32.53	34.79	38.08	1.96
4.0	36.62	39.69	43.43	2.56
4.5	41.34	45.74	48.62	1.96
5.0	48.07	49.64	51.51	0.98
5.5	52.86	54.23	56.88	1.18
6.0	57.5	60.12	62.65	1.76
6.5	60.79	64.33	68.89	2.27
7.0	68.89	69.88	70.94	0.66
7.5	69.58	74.74	80.77	4.16
8.0	77.33	79.99	83.43	2.03
8.5	77.47	83.77	92.24	5.05
9.0	87.06	90.06	92.98	1.85
9.5	90.64	95.26	98.39	2.80
10.0	91.11	99.35	108.1	5.42

Table 2.6: Time required for resource encryption

resource; as such Table 2.6 reports the minimum and the maximum time recorded during the experiment, along with the time consumption average and the standard deviation.

The results of the experiments collected in this Section gave us encouraging feedbacks about the presented architecture. Indeed, thanks to the data collected through SocialCloudShare, we can state that the presented architecture is able to enhance security and privacy in Social Networks, without affecting users' experience. As such, The decentralized architecture presented in this chapter have been our reference model for all the researches developed after this work.

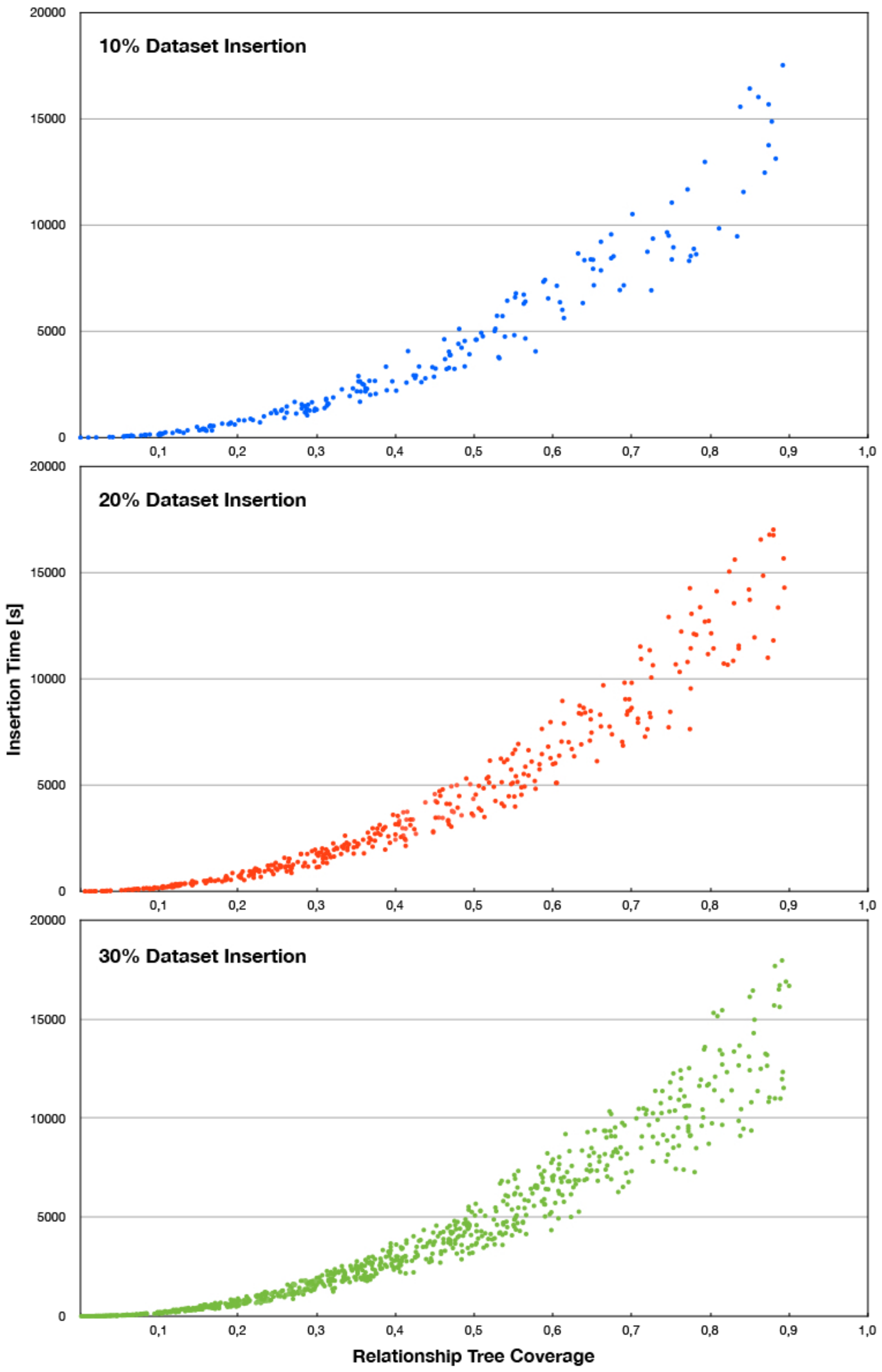


Figure 2.13: Time for new user insertion based on relationships coverage

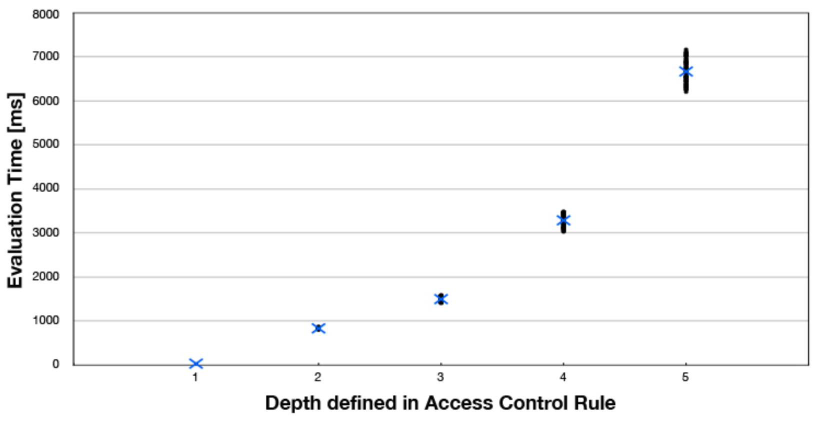


Figure 2.14: Access Control Rules Evaluation, by varying depth

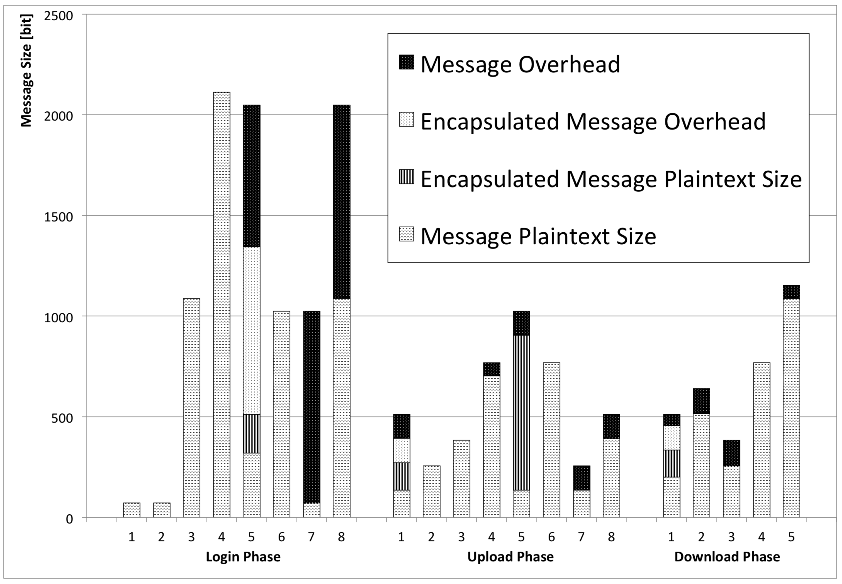


Figure 2.15: messages Encryption Size Overhead

3 Privacy Settings Recommender

As previously mentioned in Section 1.1, we are aware that many of the most popular OSN providers do not implement in their platforms access control models fully compliant with RelBAC, as properly emphasized in [19]. In some cases fine-grained configurations are supported by giving users the possibility to define customized lists of contacts to which assign access privileges. In other cases, though, users are only asked to choice whether a resource should be *private* or *public* (i.e., released to anyone in the social network). This fact, thus, limits the users' capability to define access control policies such that these reflect their own attitude towards privacy.

Given the encouraging results collected from the architecture presented in Chapter 2, we decided to continue our researches on privacy and security in DSN scenarios. As such, for what concerns the above introduced issues, we have been researching over the definition of a technique capable of helping DSN users in defining RelBAC policies that fully reflects their own attitude towards privacy.
As underlined in [39, 50], average users have difficulties in properly setting their RelBAC policies, delegating to the social network provider the task to handle automatically their configurations by exploiting pre-defined policies. Then, we found it necessary to investigate the definition of tools that help users in configuring their own privacy settings, limiting the possibility that misconfigured RelBAC policies lead to harmful data release and expose the privacy of others.

Thus, we believe it is necessary to define a tool that guides DSN users in the definition of their RelBAC policies, offering customized suggestions tailored on each user's attitude towards his/hers own privacy.
Unfortunately, the definition of a RelBAC policy is influenced by many factors. Indeed, each user has a different attitude towards his/hers own privacy and this influences the RelBAC policies that he/she defines. Thus, to define a technology capable of helping users in their RelBAC policies definition, it is required to learn and model each user's privacy attitude (i.e., his/hers attitude towards his/hers own privacy), so as to exploit the learned preferences to define precise and customized RelBAC policies. At first, then, we decided to denote as *"privacy preferences"* all the subjective variables that influence a users' RelBAC policy definition. A privacy preference, thus, represents a correlation between certain properties that data may have (i.e., those properties that make the user decide if it is sensitive data or not) and the RelBAC policy that the considered user is used to assigned to any resources that show these data properties.

Then, we have been working on the definition a privacy settings recommender capable of learning users' privacy preferences so as to exploit them so as to suggest DSN users customized RelBAC policies, tailored on their own habits. With more details, we designed a recommendation system such that, given a certain DSN user, at first it learns the correlation that exists between a resource properties and the RelBAC policy that he/she is used to define for sharing resources with that property values. Then, each time a DSN user wants to upload a new resource into the platform, the recommender exploits the learned correlations to select the policies that are related to topics of the new resource. Finally, it generates a new access control policy by combining the retrieved ones, returning to the considered user as a customized suggestion.

According to our initial expectations, we initially designed the above introduced recommendation service as part of the architecture introduced in Chapter 2. As such, the recommender tool described in this chapter was meant to be as an additional service inside the RMS, which exploited the CS as a front-end interface with which the user interaction was realized. Although, we then decided to pursue the declaration of the presented technique for any social network service. As such, in the remainder of this Chapter, we will denote with OSN any *online social network* service, without making a distinction between centralized or federated architectures.

3.1 Reference Model

Despite most of the fundamental reference concepts have already been presented in 2.1, we still need to introduce *association rules*, since they play a fundamental role in the development of our recommender system.

3.1.1 Association rules

Association rules, as firstly defined by Agrawal *et al.* [4], are another key-feature for the work gathered in this chapter. As such, given a set of items $I = \{i_1, i_2, \dots, i_n\}$, a set $S = \{s_1, s_2, \dots, s_n\}$ of subsets of I (i.e., $s_i \subseteq I$), and two sets X, Y such that $X,\ Y \subseteq I$, $X \cap Y = \emptyset$, an association rule $X \Rightarrow Y$ is an implication which points out that, each time a certain s_i includes X (i.e., s_i contains the items that are contained in X), then $Y \subset s_i$ as well. In the remainder of this chapter we will then denote the association rule itemsets X and Y, respectively, as the left-hand-side (LHS) and the right-hand-side (RHS) of the rule. More details concerning association rules and the respective learning process are gathered in Section 3.2.

3.2 Learning Privacy Preferences

As a first step, we need first to formalize the concept of user's privacy preferences, as these will be the reference structure over which performing the data mining process. We recall that privacy preferences aim at modeling the users's preferences in terms of how his resources have to be shared in OSN and that, in general, information sharing is regulated according access control policies specified by grantor (data owner). As such, privacy preferences could be modeled by the same formalism adopted for access control policy in OSN (see Definition 2.1.1). However, we have to keep it in account that we expect user's preferences depend on resource's properties. Indeed, learning on raw access control policies, where resources are simply identified by their ids, would do not allow to learn dependence between resource properties and privacy preferences. As such, we present here the reference model for privacy preferences, which comes from a slight modification to Definition 2.1.1.

Definition 3.2.1 *Let u be an OSN user, let $\mathcal{ACP}$ be an access control policies catalog, and let $\mathcal{ACP}_u$ be the catalog of the access control policies defined by u, that is*

$$\mathcal{ACP}_u = \{acp \in \mathcal{ACP} \mid acp.\texttt{Grt} = u\}.$$

Then, the set of the privacy preferences of user u, which is denoted as $\mathcal{PP}_u$, contains, for each acp in $\mathcal{ACP}_u$, a privacy preference pp in the following form

$$pp = \langle acp.\texttt{Subject}, des(acp.\texttt{Object}), acp.\texttt{Sign}\rangle^1 \tag{3.1}$$

[1] We recall that the policy component represented by $acp.\texttt{Per}$ is given as implicit.

where $des(acp.\texttt{Object})$ is a function which takes as input an access control policy object (i.e., a resource) and produces as output a finite list of resource descriptors.[2]

As such, the difference between Definition 2.1.1 to Definition 3.2.1 consists in that access control policy objects are mapped to a finite space of labels, e.g. resource descriptors, which describe and represent the resource type (e.g., text, picture, music) and the resource content.

Example 3.2.1 *Let $\mathcal{ACP}_{Bob}$ a catalog containing access control policies defined by Bob, according to the ones presented in Example2.1.1. Then, $\mathcal{PP}_{Bob}$ contains the following privacy preferences:*

$$pp_1 = \langle\langle \text{``}friends\text{''}, 0.6, 1\rangle, \{\texttt{Picture}, \texttt{Seaside}, \texttt{Sun}, \texttt{Boat}\}, \oplus\rangle \tag{3.2}$$

$$pp_2 = \langle\langle \text{``}colleagues\text{''}, 0.5, 1\rangle, \{\texttt{Picture}, \texttt{Ballroom}, \texttt{Balloon}\}, \ominus\rangle \tag{3.3}$$

$$pp_3 = \langle\langle \text{``}relative\text{''}, 0.7, 1\rangle, \{\texttt{Picture}, \texttt{Family}, \texttt{Cake}, \texttt{Party}\}, \oplus\rangle \tag{3.4}$$

where the resources descriptors represent properties of 3 distinct pictures.

From Definitions 2.1.1 and 3.2.1 we recognize that the privacy preferences of a certain user consists of the access control policies that he/her has defined in time. More precisely, assuming that all the access control policies defined in the OSN platform are stored in a catalog denoted as $\mathcal{ACP}$, we can project the set $\mathcal{ACP}_u$ as the set of the access control policies defined by a given user u. This set represents the u's privacy preferences. Indeed, by extracting the resource descriptors from each object of the policies collected in $\mathcal{ACP}_u$, we can turn those policies into u's privacy preferences. We denote this set as $\mathcal{PP}_u$.
It is reasonably clear that our target, then, is to search inside the catalog $\mathcal{PP}_u$ for patterns that represent users habits for what concerns their own privacy, so as to exploit those patterns while composing new and customized access control policies.

3.2.1 Learning Process

Now that is clear that the target of this proposal is to exploit association rules while analyzing privacy preferences is order to learn users' habits, more details about the data mining process can be given.
It is relevant to note that we are not interested in learning any kind of association rule, but only those which lead to a `Subject` component, or to one of its sub-components, as

[2]We assume that with each resource is associated a set of *resource descriptors*, describing meaningful properties, like content type, data of creations, and most importantly content topics. As such, we assume that the recommender presented in this chapter is coupled with content analysis tools, such as IBM AlchemyAPI or Google Cloud Vision API (https://cloud.google.com/vision/)

the following example clarifies.
Indeed we are interested to learn, given a certain user u and a certain resource r , how he/she is used to share over the OSN that very resource, that is, we are interested to learn the implication that underlies between the couple (u, r) and the constraints that u poses over the sharing of r.

Example 3.2.2 *Let $\mathcal{PP}_{Bob}$ the privacy preferences catalog introduced in Example 3.2.1. Analyzing all the privacy preferences in $\mathcal{PP}_{Bob}$ the following association rules are detected:*

$$\{\texttt{Picture}, \texttt{Seaside}, \texttt{Boat}\} \rightarrow \text{“}friends\text{”} \tag{3.5}$$

$$\{\texttt{Picture}, \texttt{Family}\} \rightarrow \text{“}relative\text{”} \tag{3.6}$$

$$\{\texttt{Picture}\} \rightarrow \oplus \tag{3.7}$$

$$\{\texttt{Cake}\} \rightarrow \{\texttt{Party}\} \tag{3.8}$$

It is important to note that association rules 3.5, 3.6, or 3.7 lead to information which is significant in the recommendation process, whereas rule 3.8 is useless to our purpose, since it gathers no information to be exploited by the proposed recommender.

For what concerns the association rule mining process, we decided to exploit Apriori algorithm [4], which is one of the most common algorithm for what concern the association rule mining. Although we are aware that such algorithm suffers of some limitations and many other algorithms, such as Eclat [72] or FP-Growth [41], could be exploited instead of Apriori algorithm, in this Chapter we preferred to adopt the simpler one, postponing as future work a detailed comparison with other algorithms. The choice to exploit the Apriori algorithm instead of other solutions has been mainly given by the fact that the Apriori algorithm can be seen as the execution of two sub-procedures that run one after the other, as properly described in details in Sections 3.2.2 and 3.2.3. The first is the *frequent itemset lookup* procedure that analyzes the input catalog searching for tuples of items (i.e., the itemsets) that occur with a significant frequency. The second is the *association rules extraction*, that analyzes the detected itemsets and verifies the presence of association rules among them. By separating the two sub-procedures one from the other we have been able to design an optimized strategy in which each procedure is run only when necessary, avoiding an unnecessary resources (i.e., time and memory) consumption. Moreover, by separating the frequent itemset lookup procedure from the association rules extraction phase, we have been able to treat the information learnt by each user separately keeping the possibility, at the same time, to rejoin all the data together if necessary.
As an example for what concerns the two sub-routines of the Apriori algorithm, while

analyzing $\mathcal{PP}_{Bob}$, the frequent itemset lookup procedure would probably detect the following tuples with a relevant frequency

$$\{\texttt{Picture}, \texttt{Seaside}, \texttt{Boat}, \text{“}friends\text{”}\}$$
$$\{\texttt{Picture}, \texttt{Seaside}, \texttt{Boat}\}$$

since association rule 3.5 in Example 3.2.2 is deduced from them, whereas itemsets like $\{\texttt{Picture}, \texttt{Seaside}, \texttt{Boat}\}$ or $\{\texttt{Boat}, \texttt{0.6}\}$ appear in $\mathcal{PP}_u$ with lower frequency, bringing to deduce no association rule from them.

In our proposal, we consider the frequent itemset lookup and association rules extraction sub-procedures as two distinct algorithms, so as to store in a repository all the information returned by the *itemset lookup* phase, and to run the association rule extraction routine each time new data is available, updating the frequencies of the found itemsets. This modification gives us the benefits of storing any found itemset, and not only those with a remarkable frequency, as the original algorithm does. As such, when new access control policies are defined and, then, new privacy preferences are available, the catalog of the found itemsets can be easily and quickly updated, with no need to compute all of them from scratch.
Indeed, given a certain privacy preference *pp* computed from a newly defined *acp*, with this modification it is possible to increase by one the frequency of all the itemsets extracted from *pp*.
Nevertheless, we can state that, despite our modification about the execution of the lookup phase, the behavior of the Apriori algorithm is not changed. Thus, before starting the association rule extraction procedure, a threshold on a minimum frequency value is applied so as to skim the unfrequent itemsets and run the mining process only on selected itemset. The *association rule* mining process is then executed over those extracted itemsets only when necessary, that is, only when is necessary to extract new implication, so as to limit the computational impact of the procedures on the user experience.
In the following we discuss both the phases.

3.2.2 Itemsets lookup

Let u be an OSN user and let $\mathcal{PP}_u$ be the catalog of the privacy preferences of u. For each $pp_i \in \mathcal{PP}_u$, we transform the privacy preference into a set, denoted as I_{pp_i}, of atomic items, one for each *pp* component. More precisely, the *pp* elements $acp.\texttt{Sign}$ and $acp.\texttt{Sbj}$ are melted together, so as to be able to recognize a negative constraint from a positive constraint even in a de-structured privacy preference.

Example 3.2.3 *Let $\mathcal{PP}_{Bob}$ the privacy preferences catalog introduced in Example 3.2.1. The itemsets extracted from each of these privacy preferences are defined as follows:*

$$I_1 = \{\texttt{Picture}, \texttt{Seaside}, \texttt{Sun}, \texttt{Boat}, \oplus friends, \oplus 0.6, \oplus 1\}$$
$$I_2 = \{\texttt{Picture}, \texttt{Ballroom}, \texttt{Balloon}, \ominus colleagues, \ominus 0.5, \ominus 1\}$$
$$I_3 = \{\texttt{Picture}, \texttt{Family}, \texttt{Cake}, \texttt{Party}, \oplus relative, \oplus 0.7, \oplus 1\}$$

When the association rules relative to a given user u are not known, a first lookup procedure is performed over $\mathcal{PP}_u$. The procedure, then, computes a new catalog $\mathcal{I}_u$ that contains all the itemsets that can be extracted from the preferences, along with the frequency with which each itemset is found. Those frequency values, thus, become the *support* value of the corresponding itemset.

Example 3.2.4 *Let $\mathcal{PP}_{Bob}$ the privacy preferences catalog introduced in Example 3.2.1, and I_1, I_2, and I_3 the itemsets introduced in Example 3.2.3. The itemset catalog $\mathcal{I}_{Bob}$ will contain, for what concerns the 3 above mentioned itemsets, an entry for any possible combination of the items contained in each itemset. As such, the catalog $\mathcal{I}_{Bob}$ would look like follows:*

$\{\texttt{Picture}\}$	*3*
$\{\texttt{Picture}, \oplus 1\}$	*2*
$\{\texttt{Picture}, \texttt{Seaside}\}$	*1*
$\{\texttt{Picture}, \texttt{Seaside}, \texttt{Sun}\}$	*1*
$\{\texttt{Picture}, \texttt{Seaside}, \texttt{Sun}, \texttt{Boat}\}$	*1*
...	

The itemset catalog $\mathcal{I}_u$ can be updated any time the user defines a new *acp* without starting from scratch every time. As such, we can state that, given a certain user u, the catalog $\mathcal{I}_u$ dynamically grows with the number of access control policy that u defines, keeping updated all the support values respectively to the several itemsets.

3.2.3 Association Rule Extraction

Regarding the association rule extraction, our proposal does not deviate significantly from the standard Apriori algorithm. As such, when the association rule extraction procedure has to run, that is, when there is necessity to detect new rules or update previously found rules, the following steps are executed.
At first, given a user u and the corresponding catalog of itemsets $\mathcal{I}_u$, the procedure prunes the catalog so as to ignore all those itemsets which support is less than a minimum support threshold δ.[3] It is important to note that the itemsets whose support is below

[3]The minimum support threshold value δ has to be defined, trivially, prior to the execution of the procedure. It can be defined empirically evaluating the execution results.

the threshold are not deleted, since the catalog $\mathcal{I}_u$ can be updated and, then, it may occur that these support value pass the threshold.
For each itemset X in $\mathcal{I}_u$, the procedure takes into account, one per time, all the other itemsets Z such that $X \subseteq Z$. As an example, given the itemset X={`Picture`, `Seaside`, `Boat`}, the association rule extraction procedure searches in $\mathcal{I}_u$ all those itemsets such that they contain X, like, as example, Z={`Picture`, `Seaside`, `Boat`, "friends"}. Then, for each couple (X, Z) the procedure computes the association rule $X \Rightarrow Y$, where $Y = Z \setminus X$ along with its confidence value $conf(X \Rightarrow Y)$, which is computed as the ratio between the support of the $X \cup Y = Z$ itemset and the support of X itemset. In association rule mining processes, the confidence value of an association rule represents how much the considered rule can be taken into account as a meaningful rule, and gives a measure of the importance of the rule. As example, a confidence value equals to 1 means that any time the left part of a rule (i.e., the itemset X) is contained in larger itemsets, then also the right part (i.e., itemset Y) is present in those itemsets; a confidence value equals to 0.5, though, means that the right part occurs only in halves of the occurrences of the left part. With more details, then, the confidence value is computed as follows

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} = \frac{supp(Z)}{supp(X)}. \tag{3.9}$$

As such, assuming that X={`Picture`, `Seaside`, `Boat`} appears in $\mathcal{I}_u$ with a support value of 9 and assuming that Z={`Picture`, `Seaside`, `Boat`, "*friends*"} has a support value of 7, the association rule resulting from the couple (X, Z) would be {`Picture`, `Seaside`, `Boat`} $\rightarrow$ "*friends*".
Each time a new rule is found, the rule is returned along with its confidence value, unless the confidence value is less than a minimum confidence value threshold ϵ. As such, we can state that the procedure returns a catalog $\mathcal{AR}_u$ of all the association rules, along with their confidence value, corresponding to the habits of user u. Differently from the itemset catalog, though, $\mathcal{AR}_u$ is returned by this procedure is deleted prior to any execution of the procedure, in that support values that changes from one run to another cause that confidence value change as well, but also may bring to detect rules that were not previously found. We have to remark that in our proposal, we are interested only those association rules in $\mathcal{AR}_u$ for which the right part contains an *acp*.`Sbj`, or one of its components at least.

Example 3.2.5 *Let $\mathcal{AR}_{Bob}$ the catalog gathering the association rules detected for OSN user Bob. $\mathcal{AR}_{Bob}$ contains, among other things, the association rules previously presented in Example 3.2.2, and can be represented as follows:*

$\{\texttt{Picture}, \texttt{Seaside}, \texttt{Boat}\} \rightarrow$ "*friends*"	$0.7\bar{7}$
$\{\texttt{Picture}, \texttt{Family}\} \rightarrow$ "*relative*"	0.63
$\{\texttt{Picture}\} \rightarrow \oplus$	0.65
$\{\texttt{Picture}\} \rightarrow \ominus$	0.35
$\{\texttt{Cake}\} \rightarrow \{\texttt{Party}\}$	0.89
$\{\texttt{Picture}, \texttt{Family}, \texttt{Cake}\} \rightarrow \langle$"*relative*"$, 0.7, 1\rangle$	0.72
...	

where each association rule is coupled with the corresponding confidence value. As an example, the rule $\{\texttt{Cake}\} \rightarrow \{\texttt{Party}\}$*, appears as a meaningless rule at our purpose, since the right part of the rule gathers information useless to compose an access control policy. Association rule* $\{\texttt{Picture}\} \rightarrow \ominus$*, on the other hand, appears to be meaningful at our purpose, but it will be probably be discarded due its low confidence value. Association rules like* $\{\texttt{Picture}, \texttt{Seaside}, \texttt{Boat}\} \rightarrow$ "*friends*" *or* $\{\texttt{Picture}, \texttt{Family}, \texttt{Cake}\} \rightarrow \langle$"*relative*"$, 0.7, 1\rangle$*, then, are likely to be those rules which are taken into account in the recommendation process.*
Nevertheless, it might be the case that there not exists enough association rules to generated a new complete policy subject. As example, rule 3.7 presented in Example 3.2.2 gathers no information enough so as to recommend a complete access control policy. As it will be explained in the following sections, to manage this case we propose a strategy that helps in combining values from available association rules together.

3.3 Policy Recommendation

As introduced at the beginning of this chapter, our proposal aims also at suggesting new and customized access control policies to any OSN user. As such, it is necessary to define a procedure that helps in converting the information gathered by association rules, extracted as explained in Section 3.2.3, into access control policies.

Preliminary setup phase

Given a user u and the catalog $\mathcal{AR}_u$ of the learned association rules, the suggestion process takes place every time u wishes to upload a new resource r into the Online Social Network platform. Thus, prior to execute the suggestion process, the resource r is analyzed and the set of its descriptors, denoted as $des(r)$, is extracted. Since this upload and extraction process is necessary to be executed prior to any computational process with the aim to produce an access control policy, it will be taken as implied, and in the remainder of the chapter we will prevent to repeat any reference to this preliminary phase.
Moreover, we recall that we are not interested in all the association rules that the mining

process returns, but only in those which contain, in the right part, at least a component of the access control policy subject. As such, we have interest in selecting from $\mathcal{AR}_u$ those rules for which their right part contains one or more of the components of $acp.\texttt{Sbj}$, that is $acp.\texttt{Sbj} = Sub_{ids} \mid Sub_{tuple}$. Unfortunately, though, it may happen that the rules in $\mathcal{AR}_u$ contain only one of the components that we are looking for; for this reason, then, we need to define a technique that helps us in combining the single pieces into a complete access control policy subject. This will be discussed in the following.

Policies combination-based suggestion

Since our target is to exploit association rules in $\mathcal{AR}_u$ so as to combine the information gathered from them into an access control policy, we need to define how the values of the policy component are selected. The values that have to be assigned to `Grt` and `Obj` of the new suggested access control policy, denoted as $Sacp$, are specified so as to be, respectively, a pointer to the user to who the policy is suggested and a pointer to the resource he/she is willing to upload to the OSN. Since $Sacp.\texttt{Sign}$ may have two distinct values, it is unlikely to believe that the association rules in $\mathcal{AR}_u$ would all imply a positive policy or a negative policy. It is more likely to believe, thus, that $\mathcal{AR}_u$ contains a number of rules implying a positive policy and another set of rules which imply a negative one. For this reason we expect that under some circumstances two distinct policies have to be suggested: a new positive policy, denoted as $Sacp^{\oplus}$, and a new negative policy, denoted as $Sacp^{\ominus}$.
For what concerns the values for `Sbj` of these two new policies, though, we have to note that the component may have two forms: the first is the one which directly includes the list Sub_{ids} of the users for whom the policy have been defined; the second one contains the values of the tuple Sub_{tuple}. As such, returning two access control policies may not be enough, in that the association rules may lead to infer the values for both Sub_{ids} and Sub_{tuple}. As such, we will have to return up to 4 distinct suggested access control policies, respectively denoted as $Sacp^{\oplus}_{Sub_{tuple}}$, $Sacp^{\ominus}_{Sub_{tuple}}$, $Sacp^{\oplus}_{Sub_{ids}}$, and $Sacp^{\ominus}_{Sub_{ids}}$, where the first two policies will have the `Subject` component made up of the Sub_{tuple} constraints, whereas the latter will have the `Subject` component containing the Sub_{ids} users list.
As such, the suggestion process starts by taking as input the catalog of the association rules produced by the learning process, that is $\mathcal{AR}_u$. The first step done by the procedure consists in looking inside $\mathcal{AR}_u$ for all the rules for which the right part of the rule is one, or more, pointer to other OSN users. Then, if those pointers have the $\oplus$ prefix, such pointers are added to $Sacp^{\oplus}_{Sbj_{ids}}.\texttt{Sbj}$; otherwise if the prefix to the pointer is $\ominus$ those are added to $Sacp^{\ominus}_{Sbj_{ids}}.\texttt{Sbj}$. In order to complete the values for $Sacp^{\oplus}_{Sub_{tuple}}.\texttt{Sbj}$ and $Sacp^{\ominus}_{Sub_{tuple}}.\texttt{Sbj}$, though, a more sophisticated procedure is necessary. For the explana-

tion of such procedure, though, we will not take into account the difference between the two above-mentioned policies but we will consider a generic suggested policy *Sacp*; the extension of the procedure to consider the positive and negative policies is then implied. Thus, the procedure starts by initializing a local variable s as an empty $Sacp.\texttt{Sbj}$, that is $s = \langle \perp, \perp, \perp \rangle$. At first, then, we consider those association rules for which their right part contains the complete triplet of values. If more than one triplets of values are available, only the more restrictive triplet is considered, that is the one that realizes the smallest set of OSN users.[4] If no association rule implies a complete triplet of value, the procedure starts in analyzing those rules for which their right part contains 2 values out of 3 of the triplet. Again, the couples of values are sorted, and the one with the more restrictive values is selected. We have to note that this sort operation is possible in that the couples found ad this stage of the procedure are all heterogeneous. Indeed, if two distinct couples c and c' would include values for different variables, than this would imply that there must exist an association rule that implies the complete triplet of values. Although, a more detailed discussion about this scenario is gathered in next Section 3.3.
At last, since the suggested policy subject $Sacp.\texttt{Sbj}$ cannot be complete, the procedure analyzes the remaining rules, that are those which right part contain only one value. All the found values are grouped per type and then sorted from the more restrictive to the less restrictive. In case that a couple of values, if any, were stored in the local variable s, the procedure tries to complete the policy subject by adding the missing value to s. Otherwise, in case s still contains no values at this step, the procedure tries to fill the triplets with the found values.
At this point, in case the subject of *Sacp* has been completed, that is, all the 3 values of the triplet have been selected, the suggested access control policy is returned to the final user; otherwise, if some value is still missing, the procedure continues as described in the following.

Iterative power set-based suggestion

Although, it may occur that the policy suggestion procedure, as described in previous Section, does not return any suggested access control policy to the final user u. Indeed, as seen in Section 3.3, the resource to be uploaded to the OSN is firstly analyzed and its content descriptors are extracted. Then, with the set of descriptor $des(r)$, the procedure extracts from the catalog $\mathcal{AR}_u$ those rules for which their left part contains all those descriptors. Thus, if those association rules in $\mathcal{AR}_u$ do not gather enough information so as to compose an access control policy subject, no output can be produced. As such,

[4]In the complete implementation of the procedure, trivially, the more restrictive triplet is considered for the positive policy, whereas the less restrictive triplet is considered for the negative policy.

a more flexible but robust technique is necessary, so as to maximize the production of valid output.
As such, we present an iterative procedure that, each time a valid output is not produced by the suggestion procedure of Section 3.3, tries to enlarge the set of association rules passed as input by considering subsets of $des(r)$ with size decreasesing each iteration. With the power set of the set of resource descriptors denoted as $\mathcal{P}(des(r))$, and $\mathcal{P}(des(r))_{/i}$ that denotes the list of the subsets of such power set with cardinality i, the iterative suggestion algorithm is presented in Algorithm 2.

```
   Input: user $u$, resource $r$, association rules catalog $\mathcal{AR}_u$.
 1 begin
 2     var i ← $|des(r)|$;
 3     Sacp.Sbj ← $\langle \perp, \perp, \perp \rangle$;
 4     repeat
 5         var rules ← $\{ar \in \mathcal{AR}_u,\ ar.LeftPart \in \mathcal{P}(des(r))_{/i}\}$;
 6         Sacp.Sbj ← suggest(Sacp.Sbj, rules);
 7         i ← (i-1);
 8         if $\perp \notin$ Sacp.Sbj then
 9             return Sacp.Sbj
10         end
11     until $i \geq 0$;
12 end
```

Algorithm 2: Iterative suggestion procedure

The `suggest` procedure mentioned in Algorithm 2 is nothing but the policy suggestion procedure, as described in Section 3.3. As such, the `suggest` procedure has the task to analyze the input catalog of association rules and divide it into 3 groups: *(i)* those rules whose right part contains the values for the triplet of Social Graph constraints, *(ii)* those rules for which their right part contains 2 out of 3 values of the triplet, and *(iii)* those rules whose right part contains only one of the 3 values. Those groups are, then, combined as described in Section 3.3. This step aims to complete, if possible, the missing values of *Sacp*.`Sbj` with the right part of the rules passed as input, but without modifying any value previously set in *Sacp*.`Sbj`. As such, as soon as the access control policy subject does not contain empty value anymore, the suggested access control policy is returned to the final user.

Suggestion exploiting other users' experience

The technique described in Section 3.3, though, may not be enough to produce a valid output for the final user under given circumstances. Indeed, when a certain user u makes use of the procedures presented in this chapter but he/she has never put attention over his/hers privacy, it may occur that the information extracted from the association rules catalog, that is the information learnable from the already defined access control policies, is not enough to deduce the user's habits. As well, in case the users' habits are very heterogeneous, the learning process described in Section 3.2 hardly returns a catalog of association rules. Similarly, even in case a newly subscribed user wants to exploit the recommendation system presented in this Chapter, the proposed privacy recommender may not be able to return a valuable catalog of association rules. This problem, indeed, is common to any recommendation system and it is known as "cold start problem". Indeed, given a recommender that is based on previously learnt information, it is required for the system to define a set of strategies to be exploited when the background information is not available. As such, it is necessary to have a technique that helps in producing an access control policy to be suggested to the final user in case no information about his/hers habits is available.

In this chapter, then, we present two distinct techniques to be run in case the suggestion procedure, as presented in Section 3.3, does not produce a valid and complete access control policy to be suggested. The two presented techniques are both based on the definition of a support set of users, denoted as $SupportUsers$, that are computed in two different ways. The Social Network users that are included in $SupportUsers$ are exploited to select a set of association rules wider than $\mathcal{AR}_u$, so as to have more data to represents users' habits.
The first technique consists in directly ask to the involved user u to elect a set OSN user that he/she considers trusted for the topics contained in the descriptors of the uploaded resource r. Those other users are then directly inserted into $SupportUsers$. Moreover all the tagged users contained in r, if any, and the considered user u are automatically added to the set as well. As such, $SupportUsers$ contains those users that have been directly pointed out by u as trusted and reliable users.
The second alternative that we consider consists in an automated procedure that independently retrieves the users that have to be included in $SupportUsers$, given a set of user defined threshold parameters. As such, we directly ask to u to define a threshold value for the *relationship type*, the minimum *trust value*, and the maximum path *depth value*. Then, the proposed tool selects from the Social Graph those users which respects the minimum constraints posed by u and inserts them into $SupportUsers$, along with u himself.

3.4 Experimental results

In order to prove the suitability of the techniques presented in this chapter, and with the aim to evaluate their performances under multiple circumstances, we set up an experiment based on different phases. Given the difficulty to access to the background knowledge necessary for our proposal, that is, given a OSN user u, the history of all the resources that he/she published on the OSN and the respective access control policies, we set up an online platform tailored for such experiment. It is important to emphasize, though, that this does not imply that the techniques presented in this chapter are not suitable to be used in a social network; indeed, as mentioned in 3.2.2, as an example Facebook gives developer the possibility to access the list of posts (i.e., the resources) published by a given user u, along with all the details on the access control constraints posed by u. The difficulty in implementing directly this experimental prototype on Facebook, thus, comes from the necessity to convince unknown users to grant us to access, without limitations, to such data.
As such, for this experiment, we set up a small fake OSN, based on a small Social Graph containing 33 nodes (i.e., the OSN users) and 52 edges (i.e., the relationships). We considered 5 relationship types when setting up the graph, that are *friends*, *close relatives*, *relatives*, *colleagues*, and *schoolmates.*
Our experiments, then, consisted in asking a set of human evaluators to fill up a questionnaire composed as follows. At each step we asked the evaluator j to impersonate an OSN user u as the owner of the resource r, and to define, according to the social graph and to the nature of the resource, which would be the best access control policy to share such resource, according to his/hers habits.

With more details, then, we asked to 30 human evaluators, heterogeneously distributed among authors' relatives, colleagues and friends, to analyze a resource r that were shown them and to define, according to their habits, which would it be the best access control policy for r. When defining the RelBAC access control policy *acp*, each evaluator has been asked to choose a policy sign *acp*.`Sign` and to define the subject *acp*.`Sbj`, in compliance with the definition given in Definition 3.2.1. Moreover, we left evaluators the possibility to express their will to publish a certain resource as *private*, that is equal to define an access control policy that authorizes no user. At last, then, we gave evaluators the possibility to skip the evaluation due to a lack of information, so as to express a sense of uncertainty. The resources to be evaluated, at last, have been extracted from a set made up of 52 files, of which 13 pictures, 13 links, 13 Twitter-like short text posts, and 13 text posts of about 500 characters each. Even pictures and links, then, were paired with short sentences describing the resource owner's opinion on the represented topic. Despite the selection process of the resource r to be analyzed by

the evaluator j were random-based, such process have been designed so as to ensure that all the resources were evaluated the same number of times.
Then, since each evaluator was asked to defined 25 access control policies for 25 distinct resources, we had a set of about 700 access control policies to compose our background knowledge. Over this set of policies to compose the catalog $\mathcal{ACP}$, we run the second part of experiment. In the second part, we first associated each resource with a set of descriptors, so as to generate $\mathcal{PP}$. At this purpose, we generate descriptors according different strategies, with the result of different sets of $\mathcal{PP}$. In particular, a first set of resource descriptors has been generated by human using the following descriptors: $\{$*Alcohol, School, Health, Sex, Holiday, Sport, Politics, Work, Religion, None*$\}$. The resulting privacy preferences catalog is denoted as $\mathcal{PP}^\star$. Other three privacy preferences catalogs are generated by exploiting AlchemyAPI[5], that is an automatic text analyzer. As such, $\mathcal{PP}^{keyword}$, $\mathcal{PP}^{concept}$, and $\mathcal{PP}^{taxonomy}$ have been composed by replacing each resource with the descriptors returned by, respectively, AlchemyAPI keyword extractor, AlchemyAPI concept tagger and AlchemyAPI taxonomy classifier. The human-composed catalog $\mathcal{PP}^\star$ have been composed, then, so as to compare the results coming from an automatic text analyzer tool and human expertiseness.
With more details, $\mathcal{PP}^{keyword}$ have been composed exploiting the *Keyword Extraction* API[6], the $\mathcal{PP}^{concept}$ have been composed thanks to the *Concept Tagging* API[7], and $\mathcal{PP}^{taxonomy}$ have been composed exploiting the *Taxonomy* API[8]. For what concerns the $\mathcal{PP}^{taxonomy}$ catalog, actually, since the API returned labels of the form

$$\textit{super-category/category/\ldots/sub-category}$$

where the category hierarchy is clear, we decided to split the returned label, so as to represent with a label every category, without considering the hierarchy.

With these privacy preferences catalogs, then, we computed, for each evaluator j, the corresponding sets of association rules that are, respectively, $\mathcal{AR}_j^\star$, $\mathcal{AR}_j^{keyword}$, $\mathcal{AR}_j^{taxonomy}$, and $\mathcal{AR}_j^{concept}$ setting the minimum support threshold to 1[9] and the minimum confidence threshold to 75%. Then, thanks to this set of association rules catalogs, we have been able to test the suggestion procedure presented in this chapter. As such, we submitted to the policy suggestion procedure, for each evaluator j and for each resource r for which j defined an access control policy, the set of resource descriptors of r according to the extraction technique that composed the considered catalog. As exam-

[5]http://alchemyapi.com
[6]http://www.alchemyapi.com/products/alchemylanguage/keyword-extraction
[7]http://www.alchemyapi.com/products/alchemylanguage/concept-tagging
[8]http://www.alchemyapi.com/products/alchemylanguage/taxonomy
[9]In our experiment, with a base dataset composed of 25 evaluations, we had to keep this threshold to the minimum value so as to obtain remarkable outputs.

Input catalog	**without** $SupportUsers$	**with** $SupportUsers$
$\mathcal{AR}^{\star}$	36,46%	32,62%
$\mathcal{AR}^{keyword}$	53,70%	55,27%
$\mathcal{AR}^{concept}$	73,36%	75,78%
$\mathcal{AR}^{taxonomy}$	68,66%	71,65%

Table 3.1: Suggestion procedure success ratio

ple, when the suggestion procedure was set up so as to exploit the association rules in $\mathcal{AR}_j^{keyword}$, we exploit the set of descriptors $des(r)$ returned by the *keyword extraction* API as input to the procedure described in Algorithm 2.
In our experiment we decided to test the capability of our proposal to produce valid outputs, without particular remarks on its accuracy and precision, which is postponed for further investigation. This is equivalent, then, to compute the ratio between the number of produced output, i.e., the access control policies to be suggested, and the number of received input, i.e., sets of resource descriptors, as metric to evaluate the submitted task. Though, as described in Section 12, we described a procedure with the aim to produce as much valid output as possible; to achieve this task, then, we introduced the $SupportUsers$ set as a way to increase the number of available association rules, that is, to increase the size of the background knowledge necessary to our proposal in order to produce output policies. Asking human judges to select a set of trusted judges or making the procedure to compose the $SupportUsers$ set, though, would not have been significant; indeed, among the judges did not exists an exploitable Social Graph and, unfortunately, many of the judges did not even know each other in the real life. As such, we decided to exploit the whole set of judges as $SupportUsers$ set. Thus, in case the procedure was not able to produce output with $\mathcal{AR}_j^{keyword}$ as background knowledge catalog, we designed the experiment so as to exploit the whole $\mathcal{AR}^{keyword}$ catalog as background knowledge.
Given an evaluator j, then, a background knowledge catalog $\mathcal{AR}$ and a resource r, we repeated every suggestion process twice, so as to check the correctness of the output produced by the procedure.[10] Table 3.1 resumes the average results of the suggestion experiment, as described above.

After the preliminary phase described above, in which the recommender system is fed up with the information collected by human evaluator, the experiment is completed by a second phase. In this second phase each evaluator is asked to give a qualitative feedback

[10]The procedure presented in 2, indeed, is a simple procedural algorithm; as such, in case the input values to the procedure keeps unchanged, the respective output keeps unchanged as well.

over a given suggested policy. As such, given a human evaluator j and a resource r' that was not prompt to j in the first phase, we ask j to evaluate some suggested policies computed for r' and tailored over j's learnt data. Then, every evaluator is asked to give a qualitative label from 0 ("don't agree") to 5 ("agree") for any suggested policy, so as to give us the possibility to analyze the accuracy of the suggested policies.
Unfortunately, at the time this thesis have been composed, we have not been able to include the results of this second phase, since this part of the experiment is currently running.

In Table 3.1, then, we have been able only to collect a quantitative evaluation of the approach presented in this chapter. With more details, Table 3.1 collects, as percentages, the ratio between the number of times that the policy recommender system returned a policy to be suggested and the number of resources for which a suggestion have been asked.
Despite the lack of a qualitative analysis of the policy recommender system, though, from the data gathered in Table 3.1 it emerges that the presented recommendation system as remarkable potentialities and it is well worthy of further studies. Indeed, especially the datasets produced thanks to the Alchemy API, returned encouraging data, with a number of successful suggestion up to 532 on 702 requests. From Table 3.1 it even emerges that the process of categorization and description of the resources is not a trivial task that can be done superficially; indeed, an expert and precise process with a wide background ontology from which extracting descriptor labels appears to be the more suitable solution, as remarked by the results gathered with $\mathcal{AR}^{keyword}$, $\mathcal{AR}^{concept}$, and $\mathcal{AR}^{taxonomy}$. On the other hand, the results produced exploiting $\mathcal{AR}^{\star}$ proved that, even in scenario with a small set of descriptors, the presented technique is able to produce a valid output which encourages us about the validity of the presented technique in a real life scenario. Indeed, as mentioned before, many OSN users are used to only exploit default access control policy, without specifying their own privacy preferences. Even the proposal presented in Section 12, that enlarges the set of association rules by including even the ones learned from people different from the current user, emerges as a valid proposal. Indeed, despite such technique brought only small increments of about 2-4%, the rising trend of such values is encouraging. Indeed, we have to note that in our experiment the $SupportUsers$ was composed just by considering the habits learnt from other 29 judges plus the considered one. As such, when few keywords were exploited to describe the different resources, similarly to $\mathcal{PP}^{\star}$, having a larger association rules catalog brought to have a confuse dataset, from which the association rule extraction was not as successful as before, leading to have a $\mathcal{AR}^{\star}$ with less association rules than before.

Nevertheless, the results of our experiment gave encouraging results that put in evidence both the actual potential of such proposal and clarified the strategies to implement for the future developments of the presented techniques. Moreover, as introduced at the beginning of the current chapter, the presented techniques are suitable to be implemented in any social network service. As such, with reference to the decentralized architecture presented in Chapter 2, we currently plan to develop a running example of the introduced *Rule Manager Service* such that it implements the recommender system described in this chapter.

4 Enhance System Utility through Query Rewriting

In addition to the issues illustrated in Chapter 3, we are aware that misconfigured RelBAC policies may lead to harmful situations, different from the ones described in Chapter 3. As such, besides the exposure of sensitive information, misconfigured access control policies may even lead to an exacerbated data restriction that brings to a loss of utility to the OSN users. As an example, let assume that a certain OSN users authorizes a precise set of users to see the value of his/hers birth date, but he/she forgets to extends the mentioned authorized *"age"* value. Then, according to RelBAC model, none of the above mentioned users would be able to access the value of the *"age"* field, even though this information is implicitly released by releasing the birth date.

In general, thus, it emerged the necessity to define more flexibles techniques with the aim of handling misconfigured RelBAC policies, so as to limit the disclosure of situations like the one described above.
As such, we found it necessary to tackle the possibility for misconfigured policies to deny someone the access over information that he/she should be, somehow, authorized. Indeed, we believe that, whenever this occurs, the DSN provider cannot be able to fulfill requests coming from legitimate requestors, reducing the amount of information that he/she could access. Then, in order to increase the DSN users utility, in terms of resources correctly released to legitimate requestors, we decided to cope with the RelBAC misconfiguration problem with a different point of view from the one describe in Chapter 3.

As such, we investigated the possibility to define techniques capable of releasing precise information even in absence of explicit access control policies. For this reason, we found it necessary to define the concept of *"implicit authorization"* so as to denote all those information that could be safely released even when no access control policy authorizing these data is defined. With more details, we defined as implicitly authorizable all those data for which do not exists an explicit access control policy authorizing their release, but they can be harmlessly released, since they can be easily discovered nonetheless. Then, this harmless release is realized by means of the existence of implications able to extend an existing access control policy from some explicitly authorized data to the implicitly authorized data. It is important to emphasize that the concept of implicit authorization described in this Chapter does not clash with existing negative policies or with other security measures with the aim to prevent the release of sensitive information. Indeed our sole purpose is to release information for which do not exist an explicit access control policy but, at the same time, it can be retrieved by DSN users exploiting some knowledge external to the access control enforcement system.

In order to take advantage by these implications, we designed a technique capable of exploiting all the existing data dependencies (i.e., any correlation between elements) as a mean for increasing the system utility, that is, the number of queries that can be safely answered. As such, we defined a query rewriting technique capable of extending defined access control policy authorizations by exploiting data dependencies, in order to authorize unauthorized but inferable data.
More precisely, given a query q submitted by a user u requesting attributes not covered by an access control policy, the proposed extended authorization model authorizes q if the following conditions hold: *(i)* there exists one or more data dependencies (e.g., $X \rightarrow Y$) whose attributes in the determinant set (i.e. X) include all attributes requested by q, and *(ii)* there exists one or more access control policies that explicitly authorize u to access attributes specified in the determinant set of the dependencies (e.g., X). Besides the above mentioned query rewriting technique, we propose a mechanism to avoid users to infer additional non-authorized data by linking implicitly authorized attributes returned by a rewritten query. Indeed, when evaluating the implicit authorizability of certain information it is necessary to consider that even their release may lead to disclose harmful data (i.e., the correlation between distinct data). As such, to overcome this problem, we exploit an hypergraph structure to represent all possible correlations between data and to assure that the rewriting procedure does not allow any information leakage.
Along with the query rewriting algorithm, we give a formal proof of its *correctness* and *completeness*. Finally, we show how the proposed approach can indeed improve data

management systems by testing the presented technique over a real data schema and corresponding access control policies. In particular, this has been done using the Facebook data schema.

Similarly to the discussion brought introducing Chapter 3, we recognized that our developed model could be applied to any data management system, and was not suitable only for social network service providers. Moreover, as emphasized in [22], RelBAC model can be applied beyond social networking services. Then, we found it useful to enlarge our reference model beyond DSNs so as to increase the possibility for the presented techniques to be applied in a larger set of systems. As such, in the remainder of this chapter, we will introduce our proposal having a general database management system (DBMS) as a reference model.

4.1 Reference Model

As sketched out before, a generic DBMS will be exploited as a reference model for this chapter. As such, given that most of the commercial DBMS implement access control models slightly different from one another, in this chapter we will present our proposal exploiting a simplified access control model. In particular, we will refer to DBMSs enforcing a simplified discretionary access control, where an access control policy `acp` is defined as a tuple (`sbj`, `obj`, `priv`), with the semantics that any user whose identifiers are contained into `sbj` are authorized to execute privilege `priv` over the object whose name is specified in `obj`.

Furthermore, it is important to denote that the above presented model can even be seen as a simplification of the RelBAC model, as presented in Section 2.1.2. Thus, the discussion gathered in this chapter does not limit the possibilities to apply the described technique to a social network provider.

4.1.1 Data Dependencies

A relevant concept is the one of data dependency in a relational database schema. In general, a data dependency between two sets of attributes Det and Dep, denoted as $Det \rightarrow Dep$, is a constraint stating that if values of the determinant (i.e., Det) are known, then the values of the dependent (i.e., Dep) can be univocally determined. This has been investigated in the form of *functional dependencies* [12, 13], *foreign key constraints* [57] and *knowledge-based implications* [26, 40]. Differently from functional dependencies and foreign key constraints, a knowledge-based implication is a data dependency such that it is possible to uniquely determine the values of dependent attributes by knowing the values of determinant attributes plus some external knowledge, which can be easily

discovered by anyone. An example of knowledge-based implication is given by that fact that one could easily discover the zip code of a certain address when the address, the city, and the country are given.
In our proposal, we exploit indifferently these three types of dependencies. As such, we introduce a unified definition for data dependencies. This definition specifies to which relations the determinant and dependent of the dependency belong to, as these might be different (e.g., in case of foreign key constraints). In this case, the dependency definition should clearly denote how relations specified in determinant and dependent have to be combined together in order to reconstruct the view over which the data dependency holds. As such, it is necessary to make explicit an expression ψ to build the view. Moreover, it may occur that the implication represented by a dependency is verified only by a subset of the tuples of the involved relations. As such, it is necessary to specify the scope of the dependency, in the form of an expression φ selecting tuples for which the implication holds. Based on the above-mentioned considerations, we adopt the following formalization for data dependencies.

Definition 4.1.1 (Data Dependency) *Let* $\mathbf{S}$ *be a relational schema, let* R_1*,* R_n *be relations defined in* $\mathbf{S}$*. A data dependency dd between* R_1 *and* R_n *is defined as:*

$$dd : R_1.Det \xrightarrow{\mathcal{R};\psi;\varphi} R_n.Dep$$

where: $\mathcal{R} = \{R_1, \dots, R_n\}$ *is a finite and non empty set of relations in* $\mathbf{S}$*,* $R_1.Det \subseteq \texttt{schema}(R_1)$ *is the determinant set of attributes of dd,* $R_n.Dep \subseteq \texttt{schema}(R_n)$ *is the dependent set of attributes of dd,* ψ *is the expression implementing the join among relations in* $\mathcal{R}$*, and* φ *is a boolean expression denoting the dependency scope. We denote by* $\texttt{schema}(R_i)$ *the set of attributes contained in relation* R_i*.*

According to Definition 4.1.1, given the view built on the relations in $\mathcal{R}$ through the join predicates in ψ, those tuples that satisfy φ have values of attributes in $R_1.Det$ implying values of attributes $R_n.Dep$.

4.1.2 Architecture

We plan to enhance the functionalities offered by DBMSs. In particular, we refer to DBMSs enforcing discretionary access control, where an access control policy `acp` is defined as a tuple (`sbj`, `obj`, `priv`), with the semantics that users, whose identifiers are contained into `sbj`, are authorized to execute privilege `priv` (e.g., `INSERT, SELECT, UPDATE, DELETE`) over the relation/view whose name is specified in `obj`.[1] Moreover, we

[1]We assume that access control policies are stored into a unique authorization catalog, denoted as `SysAuth`, whereas we denote with $\texttt{SysAuth}_u$ all the access control policies that apply to a user u, i.e.,

Employees

<u>Eid</u>	RegNr	Name	Surname	Role	Dept
0	001978	Alice	Smith	Secretary	Accounting
1	710632	Bob	Taylor	Boss	Development
2	548235	Carl	Williams	Manager	Development
3	167459	Daisy	Jones	Secretary	Accounting
4	348612	Earl	Brown	Programmer	Development
5	783145	Frank	Wood	Programmer	Development
...					

Projects

<u>Pid</u>	ProjName	Supervisor^{Employees}
1	Hologram	2
2	LightSaber	2
3	FluxCapacitor	7
4	Tardis	7

Wages

<u>Proj</u>^{Projects}	<u>Empl</u>^{Employees}	HireDate	SalaryPerHour	SickHours	VacationHours	WorkedHours
1	1	01/03/2002	160	0	0	40
1	2	01/03/2002	100	0	0	40
1	4	01/03/2002	90	0	40	0
1	0	01/03/2002	50	8	0	32
1	5	05/10/2006	50	0	0	40
2	3	08/07/2009	35	24	0	16
...						

V_1:= SELECT $RegNr, Role, HireDate$ FROM $Employee\ e, Wages\ w$ WHERE $e.Eid{=}w.Empl$ AND $Dept$=‘Development’

V_2:= SELECT $SickHours, VacationHours$ FROM $Projects\ p, Wages\ w$ WHERE $p.Pid{=}w.Proj$ AND $ProjName$=‘Hologram’

Table 4.1: An example of relational schema

assume that DBMSs makes use of inference control monitoring tools to avoid inference attacks.

In general, literature identifies two categories of countermeasures to prevent inference attacks over database systems. The first category gathers those proposals, such as the ones presented in [15, 40], where inference channels are detected at design time. The second category groups those proposals where inference channels are monitored throughout

$\texttt{SysAuth}_u = \{\texttt{acp} \in \texttt{SysAuth} \mid id_u \in \texttt{acp.sbj}\}$, where id_u denotes the id of user u. Moreover, we use dot notation to specify the components of a tuple, that is, we denote as `acp.sbj`, `acp.obj`, and `acp.priv` respectively the subject, the object and the privilege of a given access control policy `acp`.

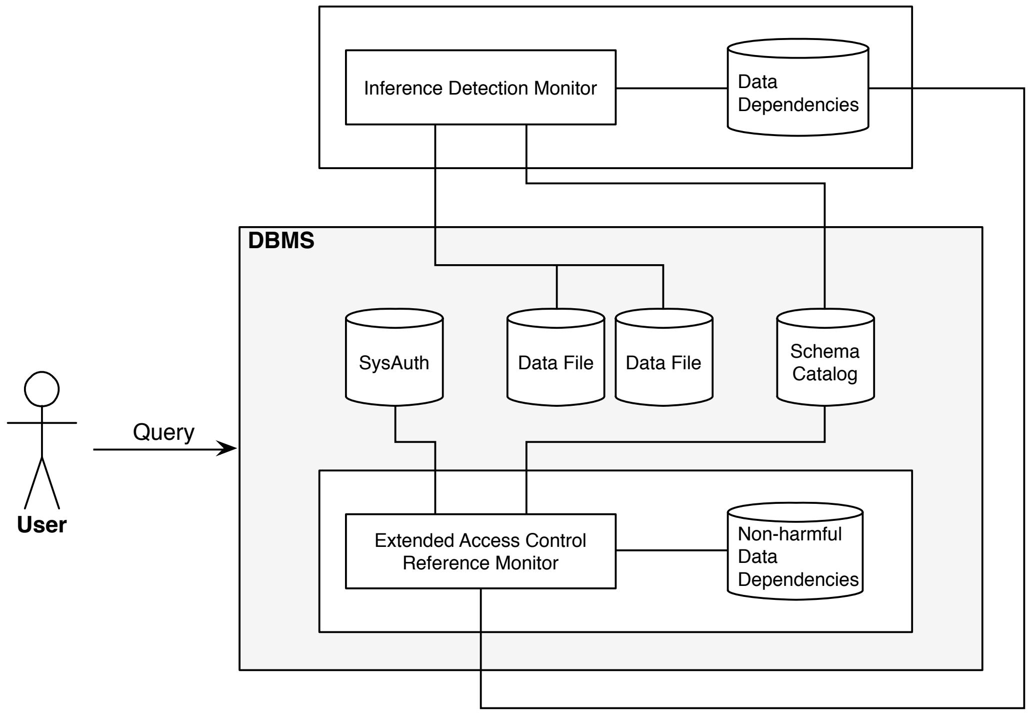

Inference Detection Monitor
Data Dependencies
DBMS
SysAuth
Data File
Data File
Schema Catalog
Query
User
Extended Access Control Reference Monitor
Non-harmful Data Dependencies

Figure 4.1: DBMS reference architecture

the database lifetime (e.g., [26, 66]). In both the cases, these proposals rely on an inference detection monitor that analyzes and finds all possible data dependencies. These dependencies are stored into a repository, as depicted in Figure 4.1. According to our solution, we assume that the security administrator checks the data dependencies found by the inference control monitoring tools, so as to select only those that can be considered as non-harmful. This task could be performed manually by the administrator, by just simply labeling as "non-harmful" all the data dependency for which their dependent set does not contain any sensitive information.[2] These labelled data dependencies are stored into a separate repository and are exploited in order to extend actual users' authorizations. In contrast, the dependencies, not specified as non harmful, are trivially considered harmful for the database system, and are used by the inference detection monitor in order to prevent sensitive data release.

[2]More sophisticated approaches could be investigated as well, aiming at automatically identifying the non-harmful dependencies. This could be reached, as an example, by exploiting security labels describing the data sensitiveness, as done in the mandatory access control model.

4.2 Implicit Authorizations

Throughout the chapter, then, we consider the following example so as to offer specific examples that clarifies our proposal.

Example 4.2.1 *Let us consider a relational schema modeling information about a company's projects. As depicted in Table 4.1, the schema consists of relations collecting information about employees and projects, and general information about employees' salaries. As reported in Table 4.1, views V_1 and V_2 are also defined. The underlined attribute names denote the relation primary keys, whereas foreign key constraints are denoted with the referenced table as superscript.*
We also assume that two access control policies, namely, $\texttt{acp}_1$ *and* $\texttt{acp}_2$ *have been specified, such that they authorize user u the* `SELECT` *privilege on V_1 and V_2, respectively. Finally, we assume that the DB security administrator has specified that the following data dependencies are non-harmful:*

$$dd_1 : Role, HireDate \xrightarrow{\mathcal{R}_1,\psi_1,\varphi_1} SalaryPerHour \tag{4.1}$$

$$dd_2 : SickHours, VacationHours \xrightarrow{\mathcal{R}_2,\psi_2,\varphi_2} WorkedHours \tag{4.2}$$

where:

$\mathcal{R}_1 := \{Employees, Wages\}, \psi_1 := Employees.Eid = Wages.Empl, \varphi_1 := Role <> \text{`}Boss\text{'}$
$\mathcal{R}_2 := \{Wages\}, \psi_2 := \texttt{true}, \varphi_2 := \texttt{true}$

Let us assume that user u submits the following query:

$$\begin{array}{l}\texttt{SELECT } HireDate, SalaryPerHour \\ \texttt{FROM } Wages\end{array} \tag{Q.1}$$

The DBMS would forbid this access due to a lack of authorizations over the Wages *relation. Nevertheless, attributes* Role *and* HireDate *are explicitly authorized by* $\texttt{acp}_1$ *and these would make u able to infer attribute* SalaryPerHour *by exploiting the data dependency 4.1.*

To overcome this limitation and improve the database system utility, in terms of successfully answered queries, this chapter presents a more flexible access control mechanism leveraging on non-harmful data dependencies. The key idea is to exploit a non-harmful dependency to extend the access rights that users have on attributes in determinant part over attributes in dependent. We refer to this extended authorization as *implicit authorization*. To this purpose, we first introduce the concept of implicitly authorizable attribute.

Definition 4.2.1 (Implicitly authorizable attribute).
Let R_i be a relation in the relational schema **S**. *The attribute* $\mathtt{a} \in \mathtt{schema}(R_i)$ *is said to be an* **implicitly authorizable attribute** *for user u if the following two conditions hold:*
(i) *there exists a non-harmful data dependency dd such that* $\mathtt{a} \in dd.Dep$*;*
(ii) *there exists an access control policy* $\mathtt{acp} \in \mathtt{SysAuth}_u$ *such that* $dd.Det \subseteq \mathtt{schema(acp.obj)}$*, where* `acp.obj` *denotes either a relation in* **S** *or a view defined over relations in* **S**.

The above definition only states whether an attribute is implicitly authorizable for a user u, but it does not specify under which conditions. Indeed, since an access control policy may have a `SQL` view as object, the policy exploited for an implicitly authorizable attribute may pose constraints on tuples over which the grant can be extended. Given a data dependency *dd*, extending the authorization holding for attributes in *dd.Det* over attributes in *dd.Dep* without considering these limitations would lead to the release of sensitive pieces of information As an example, policy $\mathtt{acp}_1$ in Example 4.2.1 authorizes the access to *RegNr, Role, HireDate* only for a subset of tuples (e.g., those with *Dept* =‘Development’). If query Q.1 was just authorized as it is, its result set would include tuples that $\mathtt{acp}_1$ does not authorize, that is, those for which the department attribute has a value different from ‘Development’. This even if *Role, HireDate* are explicitly authorized and *SalaryPerHour* is an implicitly authorizable attribute. As such, when an authorization extension is granted by exploiting data dependencies, the constraints holding for dependency determinant set of attributes have to be extended over the attributes in the dependency dependent set as well.

In order to ensure only safe implicit authorizations we propose a query rewriting procedure so as to include additional constraints imposed by access control policies exploited for the implicitly authorizable attributes. More precisely, the proposed query rewriting technique first exploits non-harmful data dependencies to compute those additional attributes, as the following example clarifies.

Example 4.2.2 *Let us assume that query Q.1 is rewritten by using data dependency* dd_1 *defined in Example 4.2.1. In this case, the constraints associated in the release of the implicitly authorizable attribute* SalaryPerHour *are computed as follows:*

$$\mathtt{SELECT}\ SalaryPerHour\ \mathtt{FROM}\ \mathcal{R}_1\ \mathtt{WHERE}\ \psi_1\ \mathtt{AND}\ \varphi_1,$$

where ψ_1 *and* φ_1 *are the constraints posed by the data dependency (see Definition 4.1.1). Then, query rewriting procedure adds further constraints in the where clause to imple-*

ment acp$_1$*. Thus, query Q.1 is rewritten as follows:*

$$\begin{array}{l}
\texttt{SELECT } HireDate, SalaryPerHour \\
\texttt{FROM } Wages \\
\texttt{WHERE } SalaryPerHour \texttt{ IN (} \\
\quad \texttt{SELECT } SalaryPerHour \texttt{ FROM } \mathcal{R}_1 \\
\quad \texttt{WHERE } \psi_1 \texttt{ AND } \varphi_1 \\
\quad \texttt{AND (}Role, HireDate\texttt{) IN (} \\
\qquad \texttt{SELECT } Role, HireDate \texttt{ FROM } V_1 \\
\texttt{))}
\end{array} \tag{Q.2}$$

Query Q.2 assures that Role and HireDate attributes are selected out of V_1, which is the authorized object of acp$_1$*. The inner query ensures that only those values of dd_1.Det (i.e., Role, HireDate) that are explicitly authorized by* acp$_1$ *are returned.*

Let us assume that the rewritten query Q.2 is executed over the schema depicted in Table 4.1. The nested query implementing acp$_1$ *(i.e.,* SELECT *Role, HireDate* FROM *V_1) returns the following result set:*

Role	*HireDate*
Boss	*01/03/2002*
Manager	*01/03/2002*
Programmer	*01/03/2002*
Programmer	*05/10/2006*

where only information of employees in the "Development" department is returned, as required by acp$_1$*. The outer subquery implements the data dependency over these values, with the following result set:*

Role	*HireDate*	...	*SalaryPerHour*
Manager	*01/03/2002*	...	*100*
Programmer	*01/03/2002*	...	*90*
Programmer	*05/10/2006*	...	*50*

where the tuple corresponding to the employee named 'Bob Taylor' has been removed due to the limitations contained in φ_1.

It is relevant to note that even if this rewritten query implements the acp$_1$ constraints it might bring to release unauthorized data. Indeed, there is the possibility that distinct tuples share the same value for the dependent of data dependency dd_1 (i.e., *SalaryPerHour*), with the consequence that they are all returned to the user, possibly leading to a data release that is not compliant with the defined access control policies. As an example, the execution of the rewritten query Q.2 would return the following result set:

HireDate	SalaryPerHour
01/03/2002	100
01/03/2002	90
01/03/2002	50
05/10/2006	50

where two tuples share the same value for the implicitly authorized attribute in $dd_1.Dep$ (i.e., 50) but, actually, only one of them is authorized by $\texttt{acp}_1$. This behavior leads to an inference channel, as such, a more articulated rewriting technique is necessary. In order to avoid this situation, we revise the query rewriting strategy so that the outer subquery (i.e., the query implementing the data dependency) selects tuples based on primary key values rather than $dd.Dep$ values of those tuples returned by the inner subquery (i.e., the query implementing the `acp`).

Example 4.2.3 *The following query represents an improvement of the rewritten query Q.2:*

$$
\begin{array}{l}
\texttt{SELECT } HireDate, SalaryPerHour \\
\texttt{FROM } Wages \\
\texttt{WHERE (}Proj, Empl\texttt{) IN (} \\
\quad \texttt{SELECT } Proj, Empl \texttt{ FROM } \mathcal{R}_1 \\
\quad \texttt{WHERE } \psi_1 \texttt{ AND } \varphi_1 \\
\quad \texttt{AND (}Role, HireDate\texttt{) IN (} \\
\qquad \texttt{SELECT } Role, HireDate \texttt{ FROM } V_1 \\
\texttt{))}
\end{array}
\tag{Q.3}
$$

The clause appended to the original query (i.e., query Q.1) ensures not only that the values of the attributes in $dd_1.Det$ are selected in compliance with the defined access control policy, but it assures as well that the returned values are all and only those values relative to the authorized tuples. In query Q.3 tuples selection is performed according to primary key values instead of $dd_1.Dep$ attribute values, so as to prevent the release of information that exceeds the constraints defined by the access control policies.

Without loss of generality, query Q.3 can be equivalently formulated as follows:

$$
\begin{array}{l}
\texttt{SELECT } HireDate, SalaryPerHour \\
\texttt{FROM } Wages \\
\texttt{WHERE } Wages.pk \texttt{ IN } V^\star
\end{array}
\tag{Q.4}
$$

where $Wages.pk$ denotes the primary key of $Wages$ and $V^\star$ is a `SQL` view implementing both the data dependency and the considered access control policy. More precisely, for each implicitly authorizable attribute `a`, we can define a `SQL` view denoted as $\mathbf{V}_{(dd,\mathtt{acp})}$ where dd and `acp` are, respectively, the data dependency and the access control policy exploited for the implicitly authorizable attribute `a`. $\mathbf{V}_{(dd,\mathtt{acp})}$ projects all and only the primary keys of relation R_n for those tuples authorized by `acp`, where $dd.Dep \subseteq \mathtt{schema}(R_n)$, as the following definition states.

Definition 4.2.2 (Implicit authorization view). *Let R_n be a relation in schema $\mathbf{S}$, let a be an attribute in* `schema`(R_n)*, and let u be a user. Let dd be a data dependency such that $a \in dd.Dep$ and let* `acp` *be an access control policy applied to u such that $dd.Det \subseteq$* `schema(acp.obj)`*. The primary keys of the tuples containing the* implicitly authorized values *for attribute a are gathered in a view, denoted as $\mathbf{V}_{(dd,acp)}$, defined as follows:*

$$\begin{aligned}&\texttt{SELECT } R_n.pk \texttt{ FROM } dd.\mathcal{R}\\&\texttt{WHERE } dd.\psi \texttt{ AND } dd.\varphi\\&\texttt{AND } dd.Det \texttt{ IN (SELECT } dd.Det \texttt{ FROM acp.obj)}\end{aligned} \tag{4.3}$$

where $R_n.pk$ is the primary key of relation R_n.

Example 4.2.4 *Let u be a user satisfying policy* $\mathtt{acp}_1$*, that grants the view V_1 to user u, as depicted in Table 4.1. Let dd_1 a data dependency as defined in Example 4.1. Let us assume that u submits the query q, as defined in Example 4.2.1. q is rewritten as follows:*

$$\begin{aligned}&\texttt{SELECT } HireDate, SalaryPerHour \texttt{ FROM } Wages\\&\texttt{WHERE } Wages.pk \texttt{ IN } V_{(dd_1,\mathtt{acp}_1)}\end{aligned}$$

where $V_{(dd_1,\mathtt{acp}_1)}$ is defined as follows:

$$\begin{aligned}&\texttt{SELECT } Wages.pk \texttt{ FROM } \mathcal{R}_1 \texttt{ WHERE } \psi_1 \texttt{ AND } \varphi_1\\&\texttt{AND } (Role, HireDate) \texttt{ IN (}\\&\quad\texttt{SELECT } Role, HireDate \texttt{ FROM } V_1\\&\texttt{)}\end{aligned}$$

As it will be described in Section 4.4, whenever a user u submits a query q, the proposed mechanism determines the set of implicitly authorizable attributes, if any. Then, for each of them, the query rewriting procedure inserts additional conditions in the `WHERE` clause enforcing constraints so as to select values for implicitly authorized attributes out of the corresponding view $\mathbf{V}_{(dd,\mathtt{acp})}$.

It is important to underline that, even though in some cases the authorization extension is equivalent to a new access control policy definition, in general this is not true. Indeed, assuming that a non harmful dependency $X \rightarrow Y$ is detected, the extending the existing authorizations from X to Y requires that, for each user u that is granted over X, a new policy has to be defined such that it grants u over Y as well, with the same limitations defined in the first policy. Otherwise, the existing policy that grants u to access X can be updated including Y in the policy object. Nevertheless, in a real-life scenario, this would be an error-prone task, since it requires to modify already existing, and working, elements of the access control system. In our proposal, thus, the authorization is extended only for the lifetime of a single task via input query modification, without editing the access control catalog.

4.3 Avoiding Correlations

The query rewriting approach presented in the previous section solves the problem of duplicates *dd.Dep* values. However, it does not avoid possible unauthorized attribute correlations as the following example clarifies.

Example 4.3.1 *Let assume that u submits the following query on relations depicted in Table 4.1:*

$$\begin{aligned}&\texttt{SELECT}\ SalaryPerHour, WorkedHours\\&\texttt{FROM}\ Wages\end{aligned} \tag{Q.5}$$

Moreover, let assume that along with data dependency 4.1 the schema holds also the non-harmful data dependency 4.2. This dependency states that by knowing the amount of hours that one employee has to work each week, the number of hours effectively worked can be discovered when the amount of sick hours and the amount of vacation hours are known. As such, dependency 4.2 is a knowledge-based implication, since it requires some external knowledge (i.e., the amount of hours to work per week). Since u is authorized to access $SickHours$ and $VacationHours$ by $\texttt{acp}_2$*, both attributes queried in Q.5 are implicitly authorizable attributes. Thus, according to the technique presented in Section*

4.2, the rewritten query is defined as follows:

```
SELECT $SalaryPerHour, WorkedHours$
FROM $Wages$
WHERE ($Proj, Empl$) IN (
  SELECT $Proj, Empl$ FROM $\mathcal{R}_1$ WHERE $\psi_1$ AND $\varphi_1$
  AND ($Role, HireDate$) IN (
    SELECT $Role, HireDate$ FROM $V_1$
) )                                                        (Q.6)
AND ($Proj, Empl$) IN (
  SELECT $Proj, Empl$ FROM $\mathcal{R}_2$ WHERE $\psi_2$ AND $\varphi_2$
  AND ($SickHours, VacationHours$) IN (
    SELECT $SickHours, VacationHours$ FROM $V_2$
) )
```

where the first subquery implements the data dependency 4.1 and $\texttt{acp}_1$ *whereas the second subquery implements* dd_2 *and* $\texttt{acp}_2$*.*

Query Q.6, thus, allows u to correlate two distinct attributes (i.e., SalaryPerHour and WorkedHours) that were supposed to be released according to two separate policies (i.e., $\texttt{acp}_1$ *and* $\texttt{acp}_2$*). As such, even though user u could separately compute the values for both attributes, query Q.6 discloses the correlation existing between the queried attributes. As such, the result set of query Q.6 enables user u to pair each value of SalaryPerHour with the corresponding value of WorkedHours.*

In the scope of this chapter, we denote the leakage illustrated in the previous example as *correlation leakage*, since the user who submits query Q.5 gains information about the correlation that exists between the queried attributes.

Then, it is fundamental to have the possibility to detect possible correlation leakages before a rewritten query is returned to the user. In particular, we aim at avoiding correlations among attributes: *(i)* that belong to the same relation/view; *(ii)* connected by a foreign key constraint or by a chain of foreign key constraints. In order to illustrate query rewriting avoiding such correlations, we first formally define them. In particular, in the following we denote a foreign key constraint as fk: $R_i.Rng \rightarrow R_j.Rnd$, where R_i and R_j are relations over the schema $\mathbf{S}$, $R_i.Rng$ is the referencing set of attributes, whereas $R_j.Rnd$ is the referenced set of attributes. This is equivalent to the SQL statement $R_j.Rnd$ REFERENCES $R_i.Rng$.

Inspired by [40], we exploit an hypergraph representation to model correlations among data. An hypergraph is defined as $G = (N, E)$, where N is a set of hypernodes and E is a set of hyperedges, which are defined as non-empty subsets of N. In general, a correlation between two given nodes $n, n' \in N$ can be modeled with an hyperpath. Formally, given an hypergraph G and a couple of nodes $n, n' \in N$, we say there exists an hyperpath $hpath(n, n')$ between n and n' if there exists a finite ordered list of hyperedges $\mathcal{L}$=$[e_1, \dots\ e_m]$, $e_i \in E, i \in [1..m]$ such that: *(i)* $n \in e_1 \wedge n' \in e_m$; *(ii)* $\forall e_i, e_{i+1}$, $e_i \cap e_{i+1} \neq \emptyset$, where $i \in [1, m-1]$.

Literature offers several examples of how to map a relational schema into a graph [40, 57]. Unfortunately these mappings do not fit our requirements, since they do not represent multiple-attribute keys, which play a fundamental role in certain foreign key constraints. As such, for the purpose of this chapter, we define a correlation hypergraph as follows:

Definition 4.3.1 (Correlation hypergraph). *Given a relational schema* $\mathbf{S}$*, the corresponding correlation hypergraph is a labeled hypergraph* $G_{\mathbf{S}} = \big(N, E, \Sigma, l(\cdot)\big)$*, where* N *is the set of hypernodes,* E *is the set of hyperedges,* Σ *is a set of labels, defined such that it contains the names of all relations, all views, all attributes defined in the schema, plus an additional label* "fk"*, and* $l(\cdot)$ *is a labeling function, defined such that it assigns a label* $s \in \Sigma$ *to all the hypernodes and to all the hyperedges of* $G_{\mathbf{S}}$*. With more details,*

- N *is defined such that there exists a node for each attribute in the schema* $\mathbf{S}$*, and the node is labeled with the name of the attribute itself. More formally:*

$$\forall R_i \in \mathbf{S}, \forall a \in \texttt{schema}(R_i), \exists n \in N \wedge l(n) = a.$$

In what follows, given an attribute $a \in \texttt{schema}(R_i)$*, we denote as* n_a *the corresponding node in* N*.*

- E *is defined as* $E = E_R \cup E_V \cup E_{fk} \cup E_{fk'}$*, where:*

$$\begin{aligned}
&\forall R_i \in \mathbf{S}, \exists e \in E_R \ \textit{s.t.}\ \forall a_j \in \texttt{schema}(R_i),\ n_{a_j} \in e;\\
&\forall V_i \in \mathbf{S}, \exists e \in E_V \ \textit{s.t.}\ \forall a_j \in \texttt{schema}(V_i),\ n_{a_j} \in e;\\
&\forall fk : R_i.Rng \rightarrow R_j.Rnd,\ R_i, R_j \in \mathbf{S}, \exists e \in E_{fk} \ \textit{s.t.}\\
&\qquad\qquad\qquad\qquad \forall a_k \in (R_i.Rng \cup R_j.Rnd),\ n_{a_k} \in e;\\
&\forall \mathcal{E} = \{e_1, \dots e_n\} \subseteq E_{fk} \ \textit{s.t.}\ \forall e_i, e_{i+1} \in \mathcal{E}, e_i \cap e_{i+1} \neq \emptyset,\\
&\qquad\qquad \exists e \in E_{fk'} \wedge e = \cup_{i=1}^{n}(e_i) \setminus \cup_{i=1}^{n-1}(e_i \cap e_{i+1}).
\end{aligned}$$

We consider here only the subsets of E_{fk} *whose cardinality is greater or equal than 2; as such,* $n \geq 1$*. Hyperedges in* E *are labeled such that* $l(e)$ *is the relation or the view name, in case* $e \in (E_R \cup E_V)$*, or label* "fk" *in case* $e \in (E_{fk} \cup E_{fk'})$*.*

It is important to note that $E_{fk'}$ contains those hyperedges modeling a combination of two or more foreign key constraints. This set is necessary to represent the correlation

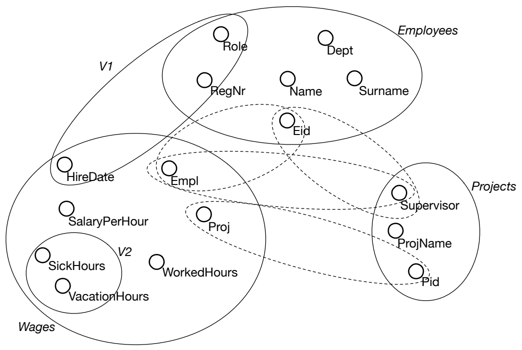


Figure 4.2: Correlation hypergraph for the schema depicted in Table 4.1

between attribute sets that are not directly connected by a foreign key constraint, but they are linked by means of two or more different constraints. For example, given a couple of foreign keys $fk : R_i.A \rightarrow R_j.B$, $fk' : R_j.B \rightarrow R_k.C$, $E_{fk'}$ contains an hyperedge e including the nodes representing attributes in $R_i.A$ and $R_k.C$.

Figure 4.2 depicts the correlation hypergraph for the relational schema presented in Table 4.1, where solid lines represent those hyperedges which are labeled with a relation or view name, whereas dashed lines represent those hyperedges that are labeled as “fk”. The correlation hypergraph is useful to detect whether a query brings to unveil a correlation, i.e., an hyperedge, that the user is not authorized to access. As such, it is necessary to identify which hyperedges of the hypergraph $G_{\mathbf{S}}$ can be actually exploited by user u. At this purpose, given a relational schema $\mathbf{S}$, we introduce an hypergraph, denoted as $G_{\mathbf{S}}(u)$, which includes all and only the elements of $G_{\mathbf{S}}$ which are actually authorized to user u.

More precisely, the **user correlation hypergraph** $G_{\mathbf{S}}(u)$ is a section hypergraph of the correlation hypergraph $G_{\mathbf{S}}$.[3] Hypergraph $G_{\mathbf{S}}(u)$ is built combining the information in $\texttt{SysAuth}_u$ with the correlation hypergraph $G_{\mathbf{S}}$. As such, for each $\texttt{acp}\in\texttt{SysAuth}_u$ and for each attribute $a\in\texttt{schema}(\texttt{acp.obj})$, the user correlation hypergraph $G_{\mathbf{S}}(u)$ contains all the nodes n_a and the hyperedges $e \in E$ such that $l(e) = \texttt{acp.obj}$. Moreover, for each hyperedge $e^* \in E$ such that $l(e^\star)=$“fk”, e^* is included in $G_{\mathbf{S}}(u)$ iff all the attributes corresponding to the nodes in e^* are authorized to u. More formally, a user correlation hypergraph is defined as follows:

Definition 4.3.2 (User correlation hypergraph). *Given a user u, a relational schema $\mathbf{S}$ and the view of the access control catalog $\texttt{SysAuth}_u$, the user correlation hy-*

[3]A section hypergraph of an hypergraph $G = (N, E)$ is defined as an hypergraph $G' = (N', E')$ such that $N' \subseteq N$ and $E' = \{e' \mid e' \subseteq N'\}$.

pergraph for u is a labeled hypergraph $G_{\mathbf{S}}(u) = (N(u), E(u), \Sigma, l(\cdot))$ defined as follows:[4]

$$(i) \qquad \forall e \in (G_{\mathbf{S}}.E_R \cup G_{\mathbf{S}}.E_V), e \in G_{\mathbf{S}}(u).E \text{ iff } \exists \texttt{acp} \in \texttt{SysAuth}_u$$

such that `acp.obj` $\equiv R_i$ *in case the object is a relation R_i of the schema or* `acp.obj` $\equiv V_i$ *in case it is a view;*

$$(ii) \qquad \forall e \in (G_{\mathbf{S}}.E_{fk} \cup G_{\mathbf{S}}.E_{fk'}), e \in G_{\mathbf{S}}(u).E \text{ iff } \forall n_{a_i} \in e, \exists \texttt{acp} \in \texttt{SysAuth}_u$$

such that $a_i \in$ `schema(acp.obj)`*;*

$$(iii) \qquad \forall n \in G_{\mathbf{S}}.N, n \in G_{\mathbf{S}}(u).N \text{ iff } \exists e \in G_{\mathbf{S}}(u).E$$

such that e contains n.

As illustrated in the next section, by using the user correlation hypergraph it is possible to verify whether attributes in a query q are connected only by hyperpaths defined in $G_{\mathbf{S}}(u)$. If this is not case, the rewritten query is not returned to the user, as a correlation leakage might occur.

Example 4.3.2 *Let u be a user and suppose that* $\texttt{SysAuth}_u$ *contains the following access control policies:*

subject	object	privilege
u	V_1	SELECT
u	$Wages$	SELECT
u	$Projects$	SELECT

A representation of the user correlation hypergraph tailored to user u over the schema in Table 4.1 is depicted in Figure 4.3. It is important to underline how the hyperedge that connects Projects.Supervisor with Wages. Empl still holds in $G_{\mathbf{S}}(u)$, even though the node which represents the referenced attribute Employees.Eid is not present. In this way $G_{\mathbf{S}}(u)$ keeps the possibility to track the existing correlation between those two attributes, that is performed by a chain of foreign key constraints.

4.3.1 Correlation Control

When u submits a query q, a *correlation control* has to be performed in order to allow to rewrite q only in case that the rewritten query q^{rw} does not lead u to infer any other data than those explicitly or implicitly authorized. This check is done by verifying the existence of hyperpaths connecting any two authorized attributes, that is, two attributes

[4]Thereafter, we use dot notation to distinguish components of $G_{\mathbf{S}}$ and $G_{\mathbf{S}}(u)$. As an example, $G_{\mathbf{S}}.E$ and $G_{\mathbf{S}}(u).E$ denotes the set of hyperedges of, respectively, the correlation hypergraph and the user correlation hypergraph.

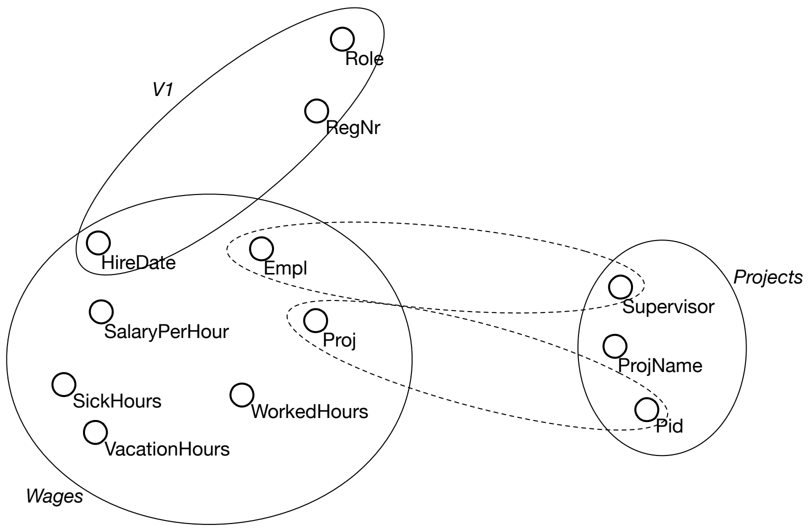


Figure 4.3: Example of user correlation hypergraph for the schema depicted in Table 4.1

for which there exists an access control policy granting u the access.
Given a SQL query "`SELECT` $\mathbf{S}_q$ `FROM` $\mathbf{F}_q$ `WHERE` $\mathbf{W}_q$ `GROUP BY` $\mathbf{G}_q$ `HAVING` $\mathbf{H}_q$ `ORDER BY` $\mathbf{O}_q$", the queried attributes are the ones defined either in $\mathbf{S}_q$ or in $\mathbf{W}_q$. As such we can define the set $requestedAttributes_q$ as the union $(\mathbf{S}_q \cup \mathbf{W}_q)$. In this set we can distinguish the attributes for which there exists an access control policy authorizing them from those which are not authorized; thus we can denote as $authorizedAttributes_q$ the set of those attributes $a \in requestedAttributes_q$ such that $\exists$ $\mathtt{acp}{\in}\mathtt{SysAuth}_u$, $a \in \mathtt{schema}(\mathtt{acp.obj})$. Therefore, given an input query q, the query rewriting procedure returns q^{rw} to user u only if, for any $(a,\ a') \in authorizedAttributes_q$, in case there exists an hyperpath connecting a and a' in $G_{\mathbf{S}}$, there exists an hyperpath connecting a and a' in $G_{\mathbf{S}}(u)$ as well. As such, we perform a correlation control defined as follows:

Definition 4.3.3 (Correlation control). *Given a query q submitted by a user u, the query is said to be correlation safe if the following holds:*

$$\begin{aligned} &\forall a, a' \in authorizedAttributes_q, \\ &\textit{if } \exists hpath(n_a, n_{a'}) \in G_{\mathbf{S}} \textit{ then } \exists hpath(n_a, n_{a'}) \in G_{\mathbf{S}}(u) \end{aligned} \tag{4.4}$$

where $hpath(n_a, n_{a'})$ is an hyperpath connecting n_a with $n_{a'}$ in the considered hypergraph.

A correlation control, as defined in Definition 4.3.3, does not need to consider implicitly authorizable attributes. As such, in case there exists an hyperpath $hpath(n_a, n_{a'})$ such that a is authorized and a' is implicitly authorizable, there must exist as well an hyperpath between a and attributes in $dd.Det$, where dd is a data dependency implying a'. As such, without loss of generality, we can consider only explicitly authorizable attributes in order to determine if a query q is correlation safe or not. Clearly, given two attributes a, a' of a relational schema $\mathbf{S}$, there may exist more than one $hpath(n_a, n_{a'})$ in the schema correlation hyperpath $G_{\mathbf{S}}$. However, we are not interested in detecting one particular hyperpath, but we only look for the existence of an hyperpath between two given nodes.

```
Data: $u$, $G_{\mathbf{S}}, G_{\mathbf{S}}(u)$, $DD$, $\mathtt{SysAuth}_u$.
Input: query $q =$ "SELECT $\mathbf{S}_q$ FROM $\mathbf{F}_q$ WHERE $\mathbf{W}_q$ GROUP BY $\mathbf{G}_q$ HAVING $\mathbf{H}_q$ ORDER BY $\mathbf{O}_q$".
begin
    $requestedAttributes_q \leftarrow (\mathbf{S}_q \cup \mathbf{W}_q)$;
    $authorizedAttributes_q \leftarrow$
        $\{x \in requestedAttributes_q \mid \exists \mathtt{acp} \in \mathtt{SysAuth}_u,\ \mathtt{acp.obj} \in \mathbf{F}_q \wedge x \in \mathtt{schema(acp.obj)}\}$;
    $requestedAttributes_q \leftarrow requestedAttributes_q \setminus authorizedAttributes_q$;
    $\mathbf{W}_{q^{rw}} \leftarrow \mathbf{W}_q$;
    foreach $T_i \in \mathbf{F}_q$ do
        if $(\{\mathtt{acp} \in \mathtt{SysAuth}_u \mid T_i \equiv \mathtt{acp.obj}\} \equiv \emptyset)$ then
            $unauthorizedAttributes_q \leftarrow requestedAttributes_q \cap \mathtt{schema}(T_i)$;
            foreach $a \in unauthorizedAttributes_q$ do
                $dependenciesForAttribute \leftarrow \{dd \in DD, dd : dd.Det \xrightarrow{\mathcal{R};\psi;\varphi} dd.Dep, a \in dd.Dep\}$;
                if $(dependenciesForAttribute \neq \emptyset)$ then
                    foreach dependency $dd \in dependenciesForAttribute$ do
                        $PoliciesForDeterminant \leftarrow \{\mathtt{acp} \in \mathtt{SysAuth}_u \mid dd.Det \subseteq \mathtt{schema(acp.obj)}\}$;
                        if $(PoliciesForDeterminant \equiv \emptyset)$ then
                            $dependenciesForAttribute \leftarrow dependenciesForAttribute \setminus \{dd\}$;
                        end
                    end
                    if $(dependenciesForAttribute \neq \emptyset)$ then
                        $dd^\star := \mathbf{pop}(dependenciesForAttribute)$;
                        $policiesForDeterminant \leftarrow \{\mathtt{acp} \in \mathtt{SysAuth}_u \mid dd^\star.Det \subseteq \mathtt{schema(acp.obj)}\}$;
                        $\mathtt{acp}^\star \leftarrow \mathbf{combinePolicies}(policiesForDeterminant)$;
                        $primaryKey \leftarrow R_j.pk,\ a \in \mathtt{schema}(R_j)$;
                        if $(\mathbf{W}_{q^{rw}} \equiv$ " " $)$ then
                            $clause \leftarrow$ "WHERE $primaryKey$ IN $V_{(dd^\star,\mathtt{acp}^\star)}$";
                        else
                            $clause \leftarrow$ "AND $primaryKey$ IN $V_{(dd^\star,\mathtt{acp}^\star)}$";
                        end
                    else
                        return unauth;
                    end
                else
                    return unauth;
                end
                $\mathbf{W}_{q^{rw}} \leftarrow \mathbf{W}_{q^{rw}} \parallel clause$;
            end
        end
    end
    $authorizedAttributes_q \leftarrow \{x \in (\mathbf{S}_q \cup \mathbf{W}_{q^{rw}}) \mid \exists \mathtt{acp} \in \mathtt{SysAuth}_u,\ \mathtt{acp.obj} \in \mathbf{F}_q \wedge x \in \mathtt{schema(acp.obj)}\}$;
    foreach $a \in authorizedAttributes_q$ do
        foreach $a' \in authorizedAttributes_q,\ a' \neq a$ do
            if (exists $hpath(n_a, n_{a'})$ in $G_{\mathbf{S}}$) then
                if (not exists $hpath(n_a, n_{a'})$ in $G_{\mathbf{S}}(u)$) then
                    return correlation_fail;
                end
            end
        end
    end
    return $q^{rw} \leftarrow$ " SELECT $\mathbf{S}_q$ FROM $\mathbf{F}_q$ WHERE $\mathbf{W}_{q^{rw}}$ GROUP BY $\mathbf{G}_q$ HAVING $\mathbf{H}_q$ ORDER BY $\mathbf{O}_q$ " ;
end
```

Algorithm 3: Query Rewriting Procedure

4.4 Query Rewriting Procedure

The query rewriting procedure as illustrated in previous sections is implemented by Algorithm 3. For simplicity, we only consider queries without any subquery. However, note that queries containing subqueries can be processed by Algorithm 3 by applying the procedure first to the inner queries, and then to the outer query. Without loss of generality, to keep the notation simple, Algorithm 3 considers only authorizations for the `SELECT` privilege.
Algorithm 3 receives as input the submitted query q, the set of access control policies which applies to the requestor user u (i.e., $\texttt{SysAuth}_u$), the set of non-harmful data dependencies DD, the correlation hypergraph $G_{\mathbf{S}}$ and the corresponding $G_{\mathbf{S}}(u)$.

As a first step, Algorithm computes the set $\mathit{requestedAttributes}_q$ (Line 2). Then, Algorithm 3 verifies which of the attributes in this set are authorized to u. In doing this, it checks which attributes in $\mathit{requestedAttributes}_q$ belong to a relation specified in the `FROM` clause $\mathbf{F}_q$ that is authorized to u, and stores them in $\mathit{authorizedAttributes}_q$ (Line 3).

Then, attributes in $\mathit{authorizedAttributes}_q$ are removed from $\mathit{requestedAttributes}_q$ (Line 4) so as to leave in $\mathit{requestedAttributes}_q$ only those attributes that are not explicitly authorized. Afterwards, the `WHERE` clause is stored into a temporary variable (Line 5). Then, for each queried table T (i.e., for each table contained in the `FROM` clause) the procedure detects all the access control policies whose object is T (Line 6). In case access to table T is not explicitly authorized by any access control policy (Line 7), the procedure computes the intersection between $\mathit{requestedAttributes}_q$ and the set of attributes of the table schema, to detect any possible implicitly authorizable attribute for T. These attributes, if any, are stored into a variable denoted as $\mathit{unauthorizedAttributes}_q$ (Line 8). Then, Algorithm 3 verifies which attributes in $\mathit{unauthorizedAttributes}_q$ are implicitly authorizable for u. Thus, for each attribute a in $\mathit{unauthorizedAttributes}_q$, Algorithm 3 computes the subset of data dependencies having a in the right part of the implication (Line 10). If no data dependency implying a is available, the algorithm returns an error message (Line 32).

If at least one dependency dd is found, Algorithm 3 starts an iteration over all the detected dependencies (Line 12). As mentioned above, the determinant of the dependency must be explicitly authorized to u, that is, there must exist an access control policy `acp` whose object is the table containing the data dependency determinant. All the eligible policies are stored into a variable named $PoliciesForDeterminant$ (Line 13). Whether no `acp` in $\texttt{SysAuth}_u$ authorizes the dependency determinant, the dependency is discarded and removed from $dependenciesForAttribute$ (Line 15).

Once all suitable dependencies have been considered, the set $dependenciesForAttributes$ contains the dependencies that implicitly make a authorizable. In case there is no data dependency in the set *dependenciesForAttributes*, the algorithm returns an error message (Line 29). Otherwise, in case that *dependenciesForAttributes* is not empty, under the assumption that there exists only one dependency that determine the iterated attribute a, the procedure denotes as $dd^\star$ the dependency contained in the set (Line 19). All the access control policies authorizing u to access the attributes in $dd^\star.Det$ are collected in *policiesForDeterminant* (Line 20). The set *policiesForDeterminant* is then passed to a function, denoted as **combinePolicies**. In case *policiesForDeterminant* contains only one access control policy `acp`, the function returns the policy itself; as such, $\texttt{acp}^\star \equiv \texttt{acp}$ (Line 21). In case more than one access control policy grants u, the access over attributes in *dd.Det*, **combinePolicies** returns a temporary access control policy $\texttt{acp}^\star$ defined such that $\texttt{acp}^\star\texttt{.sub} = u$, $\texttt{acp}^\star\texttt{.priv} = \texttt{SELECT}$, and $\texttt{acp}^\star\texttt{.obj} = V^\star$, where $V^\star$ is a temporary view defined as the union of the projection of the attributes in *dd.Det* from all the `acp.obj`. More formally,

$$V^\star = \bigcup_{\forall \texttt{acp} \in policiesForDeterminant} \big(\pi_{dd.Det}(\texttt{acp.obj})\big).$$

Here we adopt the algebraic operator $\pi_A(T)$ to denote the projection of a set of attributes A out of a table T. Once Algorithm 3 has selected both the dependency and the access control policy, it computes the view $V_{(dd^\star,\texttt{acp}^\star)}$ (see Section 4.2). Then, Algorithm 3 generates additional conditions that have to be appended in `AND` to the original `WHERE` clause, if any (Line 23).

Once all the additional constraints have been computed and appended to $\mathbf{W}_{q^{rw}}$, $authorizedAttributes_q$ is computed again, considering the attributes added by the rewriting procedure as well (Line 38). Afterwards, Algorithm 3 performs a correlation control, as described in Definition 4.3.3 (Lines 39-43). If the control fails, Algorithm 3 returns an error message. Otherwise, the input query is rewritten by replacing the previously `WHERE` clause with the newly computed $\mathbf{W}_{q^{rw}}$ (Line 48). Finally, q^{rw} is executed over the database.

4.5 Security Analysis

In this section, we illustrate the *correctness* and the *completeness* properties of Algorithm 3. According to [68], correctness ensures that for each tuple returned by q^{rw} it exists an access control policy authorizing the requestor user to gain access to it. In contrast, the completeness property ensures that each tuple returned by the original query and authorized to the requestor is also included in the result set of q^{rw}.

However, supporting implicit authorizable attributes requires to revise the traditional properties definition introduced in [68]. Moreover, it is relevant to note that, since the proposed rewriting strategy composes the rewritten query by processing at attribute level (i.e., explicitly authorized attribute and implicitly authorized attribute), the correctness property has to be proved at attribute level. Thus, for the correctness property we have to show that for each tuple t returned by q^{rw} there exists either an access control policy authorizing the requestor user to gain access to, or there exists a non-harmful data dependency and an access control policy that makes each attribute in t implicitly authorizable.

Theorem 4.5.1 (Correctness.) *Let q be the query submitted by user u, let q^{rw} be the rewritten query returned by Algorithm 3. Let $Rs(q)$ and $Rs(q^{rw})$ denote the result set returned by q and q^{rw}, respectively. Let DD be the set of non-harmful data dependencies defined in the system. For each tuple $t \in Rs(q^{rw})$ and for each attribute $a \in$ $\texttt{schema}(Rs(q^{rw}))$:*

$$\begin{aligned}&\exists \texttt{acp} \in \texttt{SysAuth}_u, \textit{such that}\\ &a \in \texttt{schema}(\texttt{acp.obj}) \wedge t[a] \in \pi_a(\texttt{acp.obj})^5\end{aligned} \tag{P.1}$$

or

$$\begin{aligned}&\exists dd^\star \in DD \wedge \exists \texttt{acp}^\star \in \texttt{SysAuth}_u, \textit{such that}\\ &dd^\star.Det \subseteq \texttt{schema}(\texttt{acp}^\star\texttt{.obj}) \wedge\\ &\wedge t[a] \in \pi_a(V_{(dd^\star,\texttt{acp}^\star\texttt{.obj})}).\end{aligned} \tag{P.2}$$

The formal proof of Theorem 4.5.1 is provided in Appendix A.

Similarly to correctness, also completeness definition has to be revised to consider the implicitly authorization concept.

Theorem 4.5.2 (Completeness). *Let q be the query submitted by user u, let q^{rw} be the rewritten query returned by Algorithm 3, where $Rs(q)$ and $Rs(q^{rw})$ denote the result set returned by q and q^{rw}, respectively. Let DD be the set of non-harmful data dependencies defined in the system. For each tuple $t \in Rs(q^{rw})$ and for each attribute a $\in \texttt{schema}(Rs(q^{rw}))$ such that:*

$$\begin{aligned}&\exists \texttt{acp} \in \texttt{SysAuth}_u, \textit{such that}\\ &a \in \texttt{schema}(\texttt{acp.obj}) \wedge t[a] \in \pi_a(\texttt{acp.obj})\end{aligned} \tag{P.1}$$

or

$$\begin{aligned}&\exists dd^\star \in DD \wedge \exists \texttt{acp}^\star \in \texttt{SysAuth}_u, \textit{such that}\\ &dd^\star.Det \subseteq \texttt{schema}(\texttt{acp}^\star\texttt{.obj}) \wedge\\ &\wedge t[a] \in \pi_a(V_{(dd^\star,\texttt{acp}^\star\texttt{.obj})})\end{aligned} \tag{P.2}$$

then $t \in Rs(q^{rw})$.

Appendix B provides the formal proof of Theorem 4.5.2.

[5] We denote as $t[a]$ the value of attribute a in the tuple t.

4.6 Truman & Non-Truman Models

The *Truman model* has been firstly presented for query rewriting techniques by Rizvi *et al.* in [60]. The truman model, as presented by authors, relies on the fact that, once a certain user query is modified due to security constraints regulating the application scenario, the outcoming result set is returned to the user without informing him/her of the query modification. As such, the user would reasonably think that the received result set gathers data retrieved from the whole database, not only from an authorized portion of it. On the other hand, this model is weak in case users can discover, from external data sources, some information about the data stored in the database; as such, for example, if a certain user is aware that a database relation contains a tuple t for which the value of an attribute a is $t[a] = 5$, a result set where that tuple t is removed due to a lack of policies is inconsistent with the user query, and the user would detect an inconsistence between the received information and the reality. For this reason Rizvi *et al.* defined the *non-Truman model* for authorization-transparent access control. In the original proposal, under the non-Truman model, a query has to pass a validity test prior to the database submission; in case the query would be compliant with user's authorization, and no modification is necessary, than the query is executed normally but rejected otherwise.

Thus, the working scenario is much more complex for our current proposal. Therefore, our rewriting process aims to extend a current authorization policy over pieces of data that are unauthorized; as such a validity test prior to query submission would imply to reject all those query that the system could rewrite. Nevertheless, the rewriting procedure presented in this chapter can operate both in compliance with the truman model with few modifications to Algorithm 3. As such, given a user u and an input query q submitted by u, the main idea is to check whether the result set coming as output from Algorithm 3, that is $Rs(q^{rw})$, contains a different set of data rather than the expected result set $Rs(q)$. In this case we have to remind that the rewriting procedure runs as a black box for users. As such, given a result set $Rs(\cdot)$, system users are not able to discover whether a submitted query q has been rewritten or not, that is, to discover whether $Rs(\cdot) \equiv Rs(q)$ or $Rs(\cdot) \equiv Rs(q^{rw})$.

At last we should consider that, given a user u and an input query q, if the received result set comes from the rewriting procedure, that is, if $Rs(q')$ is produced as output, this would imply that $Rs(q) \equiv \emptyset$ due to a lack of authorizations in case q would be processed without the presented techniques. Then, we introduce the following **validity test** for the non-Truman model.

Definition 4.6.1 (Validity test).

The validity test $vt : (user, query) \mapsto \{\texttt{true}, \texttt{false}\}$ *for the rewriting procedure described in Algorithm 3 takes as input a user and a query submitted by the user itself and produces as output a boolean value computed as follows:*

$$validity(u,q) = \begin{cases} \texttt{true} & if\ Rs(q) \equiv Rs(q^{rw}) \\ \texttt{false} & otherwise \end{cases} \tag{4.5}$$

where $Rs(q)$ *and* $Rs(q')$ *are the result sets relative to query q prior and after the rewriting procedure, both computed by a* `root-level` *super user.*

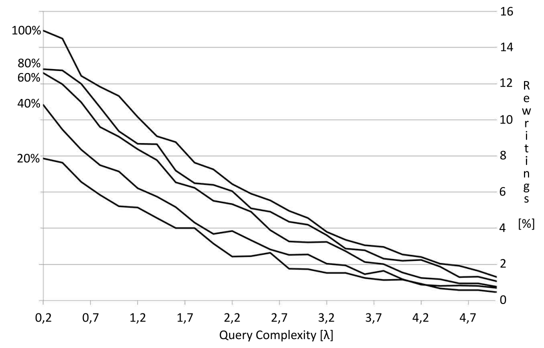
As such, given a user u and an input query q, the query q is said to be valid under the non-Truman model in case the validity test $validity(u, q) = \texttt{true}$, that is, the result set produced by the rewriting procedure $Rs(q^{rw})$ contains the same information as the result set $Rs(q)$ obtained by bypassing the authorization controls for user u and without modifying the input query. Then, for what concerns Algorithm 3, a query q is said to be valid according to the non-Truman model for user u in case the clauses introduced by rewriting procedure doesn't modify the output result set respective to the query itself. Finally, we can state that the procedure described in Section 4.4 is able to run according the non-Truman model if, when the procedure rewrites an input query q, the rewritten query q' is returned only in case $validity(u, q) = \texttt{true}$, where u is the current system user.

4.7 Experimental Evaluation

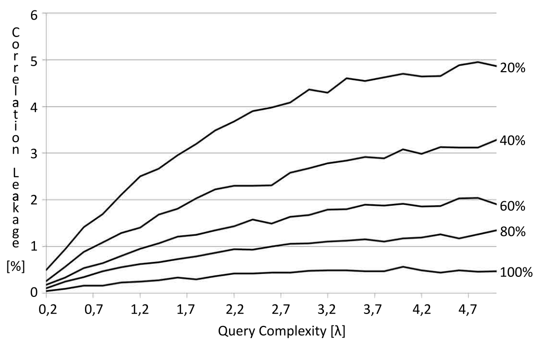
In order to prove the suitability of the presented proposal within existing DBMS scenarios, we carried out an intensive suite of tests over real database schemas. All experiments described in this section have been carried out by exploiting Java programming language and the relational DBMS MySQL. Experiments run on a MacBook Pro equipped with a 2.6 GHz Intel i7 and 8 GB of RAM. In order to be as close as possible to a real-life scenario, we select a real database schema and related access control policies as well. Thanks to authors in [8], who exploited a dataset extracted from FQL reference schema, we have been able to test our rewriting procedure on a dataset composed by both real entities and real access control policies.

The test schema is composed of 19 relations, coming from FQL entities[6], and 75 access control permissions, that represent the Facebook standard permissions over the schema entities. These have been translated into authorization on `SQL` views over the considered relations. As such, the test schema gathers tables for picture albums, application interactions, check-in into places, event participations, friendship relations, groups,

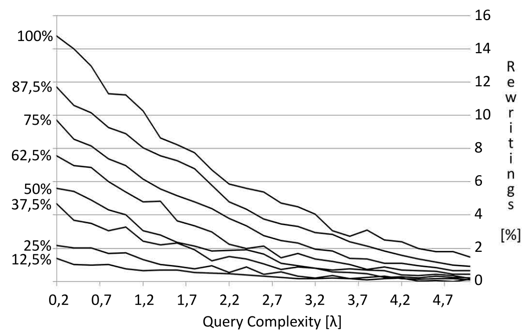
[6]Facebook Query Language:
https://developers.facebook.com/docs/reference/fql



(a) Successful rewritings



(b) Discovered correlation leakages



(c) Data dependency impact

Figure 4.4: Experimental results

text notes, videos and users data. Authorization views, such as *"a user can see all public pictures"* or *"a user can see all the pictures of his friends, where the visibility is set to 'only friends' or 'public' "* have been translated into authorization on SQL views like the followings:

```
· SELECT * FROM photo WHERE visible='everyone';
· SELECT * FROM photo WHERE visible='everyone'
    OR visible='friends' AND photo.owner IN (
      SELECT uid1 FROM friends WHERE uid2=me()
    );
```

The test users have been granted access only over these 75 views, whereas it has no authorizations on the 19 base relations.

Analyzing the FQL relational schema, we have been able to define a set of 56 data dependencies, including functional dependencies, foreign key constraints, and knowledge-based implications. The system have been tested with randomly generated queries, exploiting the query generator presented in [8]. This query generator takes as input a relational schema and a parameter λ, which regulates the generated query complexity in terms of number of selected attributes and queried tables, following a Poisson distribution. The queries are generated following a template file that contains simple queries, nested queries and correlated queries that exploit `EXISTS` operator. All tests have been carried out by varying λ in $[0.2, 5.0]$ with steps of 0.2. Given the value of λ and the number n of access control policies granted to the test users, every test has been repeated 100 times.

The first experiment concerned the output produced by Algorithm 3 by varying the number of applicable access control policies. In each round, n out of the 75 views have been randomly authorized for the test user, and the input dataset has been composed of 500 randomly generated queries. We considered only the average of the number of successful rewritings obtained at each round.

Figure 4.4a depicts the results of such experiments for what concerns the number of successfully rewritten queries out of the 500 input queries. Figure 4.4b depicts the number of correlation leakages detected by Algorithm 3. In both figures, we report the results obtained with different number of authorized policies. The graphs depict the results obtained with, respectively, 75, 60, 45, 30, and 15 authorized policies. The results are given in terms of percentages.
Comparing Figures 4.4a and 4.4b, we can note how the number of authorized access control policies have much more impact over the correlation detection than over the successfully rewritings. On the other hand, for what concerns the number of successful rewritings, the highest impact is given by the query complexity. Therefore, the higher the number of queried attribute is, the more chances we have that at least one of these attributes is neither explicitly nor implicitly authorizable.

The second experiment has been carried out in order to investigate the impact of available data dependencies over Algorithm 3. This experiment has been carried out following the same method as the first experiment; as such, the λ parameter is made vary in $[0.2, 5.0]$ and the round are repeated 100 times. Each round of this experiment has been run with a different set of data dependencies, randomly selected out of the 56 data dependencies detected in the FQL schema. Results are summarized in Figure 4.4c by considering, respectively, 56, 49, 42, 35, 28, 21, 14, and 7 data dependencies. From the graph in Figure 4.4c we can note the key role are data dependencies in the query rewriting process. As such, the more dependencies are available at runtime, the better the results are, for what concerns successful query rewritings. Even thought the best results are given on simple queries (e.g., selection of one attribute from one table), when an adequate number of data dependencies is available the presented procedure successfully rewrites about 10% out of the input queries even for non trivial queries.

Given the obtained results, we believe that our proposal may be easily implemented as part of the DBMS itself without any sensible impact over performance and users' experience. Moreover, the obtained results (see Figures 4.4a, 4.4b, and 4.4c) convince us that the impact of these techniques on a real-life scenario could significantly increase the DBMS utility, in terms of ability to successfully answer input queries.

5 Review of Literature

Decentralized Social Networks, since firstly presented in [71], have been thoroughly studied from different point of views. Nevertheless, many of the topics treated in our works were already discussed in literature. As such, this chapter gathers a comprehensive revision of the topics encountered in our research, providing dedicated sections in which any single topics is analyzed in details.

5.1 Secure information sharing in Social Networks

In recent years Decentralized Social Networks continuously grew in interest thanks to their potentialities. Indeed, the federated paradigm upon which DSN are built ensures DSN users' much more protection over sensitive information rather than centralized social networks.

Protect users' resources

Despite Tim Berners-Lee *et al.* presented in [71] only a theoretical model for DSNs, literature gathers several examples for a practical DSN implementation. Without gathering practical details concerning each implementation, among these it is important to mention Diaspora [2, 11] and OneSocialWeb [3].
Whereas Diaspora has been designed as an alternative to commercial social networks and it plans to replace them, OneSocialWeb has a wider target and it allows in its federation any kind of application. As such, one could imagine Diaspora like a decentralized

version of Facebook; on the other hand, though, OneSocialWeb implements both the classical social features (e.g., user profiles, wallboards, activity streams, etc.) and technical novelties such as fine-grained access control. Moreover, it offers any developer a suite of APIs with which any application can be integrated into OneSocialWeb.
Other remarkable examples of federated social networks are Persona [6], SafeBook [24, 23], PeerSoN [16], Vegas [29], Vis-à-Vis [61] and DECENT [44].

Given the lack of commercial interests for DSN, though, literature contains proposals tailored for centralized social networks as well. Among these we mention Scramble! [7], FaceCloak [49], Lockr [67] or Trust&Share [18]. Proposals like Scramble!, FaceCloak or Trust&Share exploit third-party storage services to replace the OSN standard storage and offer OSN users access control models that extend the one implemented by the OSN provider. As example Scramble! let OSN users define access control lists of authorised users for each piece of data, based on their preferences. Trust&Share, differently, let OSN users assign trust values to the relationships existing on the social graph and let them define access control policies based on such trust values. Similarly, FaceCloak provides a browser plug-in pushing fake information to the OSN providers and storing any sensitive data on a separated data storage.

Social graph anonymization

Differently from the above mentioned proposals, whose target is to protect users' resources such as personal profiles or UGCs, in order to protect the information gathered by the social graph requires a different approach, such as by anonymizing the social graph. The literature offers several techniques for graph anonymization, where most of them anonymize the graph by either clustering nodes (e.g., [10, 17, 42]) or graph modification (e.g., [31, 43, 48]).
However, all of them have been designed under two common assumptions. The first is that the anonymization process is carried out on the whole graph. The second consists in anonymizing the graph by modifying its topology (i.e., adding/deleting edges).
Unfortunately, these techniques cannot be applied to our scenario, since they either assume that the social graph is centralized in the social network provider or they bring to a data loss that is not acceptable for a relationship-based access control. In fact, adding a fake edge between existing nodes might create a new path satisfying some relationship-based rules, with the serious consequence of improper resource release. In contrast, deleting an edge might delete some real path making some rules no more satisfied and, as consequence, making the corresponding resource no more available to authorized users.

To the best of our knowledge the only work that assumes a decentralized scenario is the one by Terzi *et al.* [31]. Authors in [31], indeed, present a technique able to reconstruct a graph exploiting neighborhood information by exploiting a biadjacency matrix that underlies the social graph. This data structure, thus, may represent a structure with a slightly different topology from the original graph, and does not appear suitable to be coupled with a RelBAC enforcement. Furthermore it is important to mention [14] since, to the best of our knowledge, is the only work proposing a collaboration among users for computing privacy-preserving operations over graphs, without editing their topology. Unfortunately, algorithms presented in [14] are suitable only to solve path-finding problem on path of length ≤ 2.

5.2 Social privacy recommender

To the best of our knowledge, Fang and LeFevre [32] were the first to analyze the problem of privacy recommendation in a social network scenario. As such, they presented an automated wizard with the aim to assist OSN users in defining their own privacy settings, by learning OSN users' attitude towards their own privacy. By means of a survey, each user is asked to explicitly select which of his/hers contacts should see given pieces of information (e.g., date of birth, address). User's choices are then exploited to feed a binary classifier, which is capable of learning even from publicly available data (e.g., public relationships, public user's profiles). With those data, then, the wizard presented in [32] suggests either to *grant* or *deny* other OSN users to see the considered user's information.
After the proposals by Fang and LeFevre, then, many other authors followed their footsteps, proposing novel and inspiring recommender systems with the aim to help social network users in making their decisions.

Social networks recommendation tools

For what concerns social recommender many authors mainly focused on suggesting users to establish new relationships, but without focusing on the possible privacy issues that this would bring to others. Authors in [63], as example, proposed a recommendation system for signed social networks, that is, those networks in which there exists both positive and negative connections between users. As such, by labeling relationships only as *"friend"* or *"foe"*, authors analyze the social network so as to infer, and suggest, missing links with the more proper label.
Zhao *et al.* in [73] presented a remarkable proposal for what concerns social recommendation; indeed, to the best of our knowledge, authors were the first to analyze trust values and topics along with a recommender system. Authors, then, tried to solve the cold start problem by enlarging the base knowledge of a certain user with the people trusted by him/her and according the considered topics. Unfortunately, authors' attention is limited in anaylizing the social graph, without focusing on other social features.
Another notable work is gathered in [38], in which Gou *et al.* present *SFViz*, an interac-

tive tool to visualize the existing relationships that a given user has in a Social Network. All the relationships of a given users are separated and visualized per interest, that is, per community. Authors in [38] present as well a recommendation procedure that is based on the analysis of users' profile and topological similarity. Thus, the recommendation system presented in [38] suggests users new potential relationships to be estabilished in the social network.

Privacy settings recommendation tools

Munemasa *et al.*, in [52], proposed a system that recommends social network users in changing their privacy settings. As such, the recommendation system presented in [52] aims to help users in define which of their information could be visible to other users and which data should be kept as private. Unlike [32], though, authors in [52] base their learning process on a widespread analysis of *OSN* users profiles, analyzing how these users protect their information (i.e., how they set their privacy settings) based on some profile attribute values, such as age, gender or relationship status. Unfortunately, Munemasa *et al.* do not take into account to extend such process to protect other kind of resources but they only aim to protect profile information.
Shehab *et al.* [62] proposed a policy recommendation system based on an iterative semi-supervised learning approach. In this proposal, social network users are asked to manually label a set of their contacts, specifying whether these contacts should be allowed or not to see their information. Thanks to these information, then, Shehab *et al.* propose to propagate the labels throughout the social graph, so as to reduce users's efforts. Unlike [32] or other previous works, Shehab *et al.* introduce a recommendation process that exploit the properties of the social graph in the recommendation phase. Nevertheless, the techniques presented in [62] only focuses on privacy labels (e.g., "allow"/"deny") to be assigned to other users so as to authorize or not their requests.
Another remarkable proposal provided by [70], in which authors focus on a proper and secure information release concentrating on the dynamics that underlies users' location sharing privacy preferences. Indeed, [70] gathers a thoughtful analysis on how people are keen on sharing their own location. With more details, Xie *et al.* are concerned with studying how users are used to share their physical location (e.g., airport, restaurant, etc.) according to the time of the day, companion and emotion. Moreover, thanks to the results of such analysis, authors present a location sharing recommendation system with the aim to help users in sharing their location. Furthermore, the learning process in [70] analyzes the similarity between users and between scenarios (i.e., locations) and tries to extract, given a user u and a scenario s, the most accurate privacy policy.
At last we mention [58], whose authors propose an approach that turns out to be very close to our proposal. Indeed, Reinhardt *et al.* recognized that *OSN* users should be helped in defined proper access control policies for each resource they share in the

network. Authors' approach consist in analyzing the sensitiveness of the resource to be posted on the social network and the strength of the relationships in the social graph so as to combine a privacy policy to be suggested. Though, one of the limits of the proposal presented in [58] consists in recommend social network users privacy policies represented simply by list of authorized users.

5.2.1 Access control enforcement by query rewriting

Part of our researches have been related to define a query rewriting technique capable of safely extending authorizations granted by access control policies, exploiting data dependencies. These works have been extensively inspired by many proposals concerning inference channel detection.

Security models to protect data management systems from inference channels have been thoughroly investigated since early '80s by Denning[27, 28], by Thuraisingham [64, 65], and, more recently, by Bertino [9].
As highlighted by Farkas and Jajodia in [33], techniques for detecting and removing inference channels can be grouped into two main categories: *(i)* those removing inference channels at design time, and *(ii)* those that eliminate inference channels at query time.

Among the first group of proposals it is worth to mention [40], which introduces the concept of *catalytic relation* to model those inference channels arising from external knowledge. To the best of our knowledge, this paper is the first that presents an hypergraph-based model to analyze and detect inference channels. Hinke and Delugach, in [26], present *Wizard*, an automated system capable of detecting inference channels over a structured set of data. They exploit a *conceptual graph*, rather then an hypergraph, to model the inference channels.
Similarly to [40], also [26] considers to merge the provided data structures with human-supplied external information, in order to detect the more inference channels as possible.
[51] defines the concept of *sphere of influence* (SOI) as a tool to analyze all possible inferences that can arise from a given *core*, that is, a piece of information. In defining the sphere of influence, authors considers the provided data structures, user external knowledge, and additional data that may be derivable or inferrable. By means of these information, then, authors highlight any possible inference channel so as to inform any system administrator about their existence.

The second category, thus, collects those proposal that enforces checks at query time. Denning proposes in [27] a filter-based technique that acts as a proxy between the users and the data management system, avoiding any unauthorized data disclosure by means of authorization views. The proposed technique consists in comparing the result set coming from a data management filter, that returns only authorized data, and the result set coming from the database. In case the result sets are equivalent, then, the response is sent to users. [65] proposes a query modification technique aiming at preventing unauthorized data inference, considering external knowledge as well. It exploits a logic-based architecture where an inference detection engine working with an inference rules repository supports system administrator in detecting possible inference channels existing in the data management system. These above mentioned proposals analyze the structure of the submitted query and, if an inference channel is detected, the query is then rejected or modified, avoiding any unauthorized data release.

6
Conclusion

Decentralized Online Social Networks (DSNs), as firstly presented by Tim Berners-Lee *et al.* in [71], emerged as a valid and promising solution for moving users' personal data out from OSN realms. However, as properly emphasized in [11], it has been shown that DSNs suffer of given limitations that lead them to suffer of the same issues as centralized OSNs.
In this thesis we present the proposals we have developed to tackle some of the most significative problems that Decentralized Social Networks (DSNs) suffer of.

Summary

First of all, we have investigated the possibility to implement new technologies enabling DSN architecture to lay on modern cloud-based storage services in order not to burden on DSN users' offering, at the same time, a safe and secure social networking service. Then, so as to implement a cloud-based privacy-preserving information sharing service, we had to deal with many challenges.

As such, in order to let the DSN provider perform a privacy-preserving RelBAC enforcement, we have designed a collaborative anonymization technique for the social graph.
We are aware that current literature offers several techniques for graph anonymization (e.g., [10, 17, 42, 31, 43, 48]). Unfortunately, none of these fulfilled our requirements, that is, a decentralized anonymization process and the possibility to verify the existence

of a given connection between nodes without exposing users' privacy.
The presented technique allows to store into a polynomial the local view that every user has of the social graph. The composition of this polynomial enable each DSN user to represent, in an anonymized form, all of his direct contacts and, at the same time, makes it easy to verify wether a connection between two users exists or not.
Along with this anonymization technique, we have extended the current DSN architecture by means of a 4-party architecture, so as to implement a secure management of users' resources stored in the public cloud. With more details, we have decided to *(i)* encipher any user resource directly at client-side by means of a **Cipher Service** (CS) and to *(ii)* split the process of generation of the encryption keys between two separate additional entities of the extended architecture. In particular, the key generation scheme implies that only CS is able to compute the key, whereas the two entities which generates the parameters from which the encryption key is derived are not able to compute it. Moreover these two entities, which are the **Rule Manager Service** (RMS) and the **Key Manager Service** (KMS), are in charge of, respectively, storing the users' RelBAC policies and storing into the cloud-based storage service the users' resources.

After the definition of the above illustrated proposals, we have worked on the definition of novel techniques with the aim of handling misconfigured RelBAC policies.
Indeed, it is nowadays well accepted that the definition of access control policies is an error-prone task. As such, in a scenario where always more frequently personal and sensitive information is exchanged among users, misconfigured access control policies may cause to expose other users' privacy. As an example, if a certain users shares a picture setting erroneously its RelBAC policy as public, then even the privacy of every person depicted in such picture would be exposed.
Moreover, it emerged that social network users have difficulties in defining RelBAC rules that properly express their attitude towards their own privacy, as underlined in [39, 50].
For these reasons, in order to limit the RelBAC policies misconfiguration issue, we have worked on a privacy settings recommender able to assist users in defining tailored and customized RelBAC policies.
Unfortunately, since the definition of a RelBAC policy is influenced by many factors, we had to keep into account that each individual performs a different decision process to decide how a resource has to be released in DSN. In addition to this subjective aspect, we have also acknowledged that the decision an individual might take on resource release is greatly impacted by resource's contents. Based on these observations, we have designed a recommendation system such that, given a certain DSN user, at first it learns the correlation that exists between a resource properties and the RelBAC policy that he/she is used to define for sharing resources with that property values. Then, each time a DSN user wants to upload a new resource into the platform, the recommender

exploits the learned correlations to select the policies that are related to topics of the new resource. Finally, it generates a new access control policy by combining the retrieved ones, returning to the considered user as a customized suggestion.

On the other hand, misconfigured RelBAC policies may even lead to an exacerbated data restriction that brings to a loss of utility to the DSN users. As an example, let assume that a certain DSN user uploads in the network a given picture, both in low and high resolution version. In case the considered user defines a RelBAC policy authorizing, erroneously, only the hi-res version of the picture, no user would be granted to access the low-res one, even though the authorization over a larger set of information (i.e., the hi-res picture) exists.
As such, we have investigated the possibility to augment the RelBAC enforcement engine with a flexible and secure strategy capable of releasing information even when an explicit authorization are not granted.
With more details, we have first defined the concept of *"implicit authorization"* so as to denote all those information that could be safely released. As such, we have defined as implicitly authorizable all those data for which do not exists an explicit access control policy authorizing their release but they can be harmlessly released nonetheless. Thus, this harmless release is realized by means of the existence of implications able to extend an existing access control policy from some explicitly authorized data to the implicitly authorized data. Then, we have designed a technique capable of exploiting all the existing data dependencies (i.e., any correlation between elements) as a mean for increasing the system utility, that is, the number of queries that can be safely answered. As last, we have defined a query rewriting technique capable of extending defined access control policy authorizations by exploiting data dependencies, in order to authorize unauthorized but inferable data.

Future Developments

Cloud-based secure information sharing. Despite our proposed techniques have already been tested in a real life scenario by means of a Facebook application, we believe that further extensions can be developed. For instance, we plan to implement a plugin for any DSN provider so as to enable any existing DSN service to exploit our proposal.
Moreover, we plan to enhance the privacy preserving RelBAC evaluation so as to support more expressive access control rules. We also wish to enhance our studies of oblivious polynomial evaluation techniques [53], and introduce them in the collaborative graph construction process to avoid the information leakage presented in Section 2.5.2.

Privacy settings recommender. As briefly described in Section 3.2.1, we are aware that there have been proposed data mining algorithms with better performances rather than the Apriori algorithm. As such, we plan to refine the presented techniques so as to support these algorithms (e.g., Eclat, FP-Growth) so as to improve the quality of the suggestion techniques and to enhance the support techniques.
Moreover, plan to cope with the cold start problem of the recommender feeding the system with more information, as example, taken from OSN communities by exploiting techniques like the ones presented [38]. As another fundamental extension, it is required to analyze different techniques to evaluate the accuracy of the presented recommender. As such, we plan to consider a ReBAC similarity metric so as to evaluate the soundness of the recommended policies. At last, we would like to implement the presented technologies directly inside the architecture presented in Chapter 2.

Query rewriting engine. For what concerns the proposed query rewriting techniques, we plan to extend it following different directions. At first, as mentioned for our privacy settings recommender, we plan to implement a RelBAC enforcement engine that supports the presented query rewriting technique inside the plugin mentioned before.
Moreover, as highlighted by Rizvi *et al.* in [60], query rewriting techniques may return user misleading or incomplete data. As such, we plan to broaden the scope of our proposal to consider malformed access control policies so as to increase even more the intrinsic utility of the implementing system.

Then, by implementing the above introduced DSN-plugin, this would enable any current DSN provider to easily exploit the presented techniques, without any particular development issue leading, thus, DSN users to a more aware and self-conscious usage of DSN platforms.

7 Appendices

A Query Rewriting Correctness Proof

Proof: We proof Theorem 4.5.1 by showing that if $\exists t^\star \in Rs(q^{rw})$ such that $\exists a \in$ $\texttt{schema}(Rs(q^{rw}))$, for which neither P.1 nor P.2 hold, then a contradiction arises.

If P.1 does not hold, it means that there is no policy authorizing $t^\star$, thus $a^\star$ is not contained into $authorizedAttributes_q$ but it is still in $requestedAttributes_q$ computed in Line 4 of Algorithm 3. The algorithm then checks each table specified in $\mathbf{F}_q$. When the one, say $T^\star$, to which $t^\star$ belongs is considered, the **if** condition in Line 7 is satisfied. After the execution of Line 8, $unauthorizedAttributes_q$ contains $a^\star$. When $a^\star$ is considered in **for** cycle in Line 10, since P.2 does not hold, we might have two cases: (i) there exists no data dependency that implies attribute $a^\star$; (ii) there exists no data dependency that implies attribute $a^\star$ such that the dependency determinant is explicitly authorized for u.

Let us consider the first case (i), that is, that no data dependency in DD implies attribute $a^\star$. As such, the $dependenciesForAttribute$ set computed at Line 10 is empty. Thereby, the conditional statement at Line 11 is not verified and the computation goes at Line 32, where the algorithm stops and returns a message to inform that the query is not authorized. Thus, a contradiction arises.
In the second case (ii), there exists at least one data dependency $dd^\star$ with $a^\star \in dd^\star.Dep$ but there is no access control policy authorizing u to access $dd^\star.Det$. In this case, the condition in Line 11 is satisfied (i.e., $dependenciesForAttribute \neq \emptyset$), but the $PoliciesForDeterminant$ set is empty. This brings the algorithm to remove $dd^\star$ from $dependenciesForAttribute$, thus making the **if** condition in Line 18 false. As such, algorithm jumps to line 29, stops and returns an unauthorized message. Thus, a contradiction arises. □

B Query Rewriting Completeness Proof

Proof: Similarly to Theorem 4.5.1, we proof Theorem 4.5.2 by contradiction. In particular, we show that if there exists a tuple $t^\star \in Rs(q)$ such that there exists an attribute $a^\star \in \texttt{schema}(Rs(q))$ to which properties P.1 and P.2 do not apply and $t^\star \in Rs(q^{rw})$, then a contradiction arises.

If P.1 does not hold for $a^\star$, it implies that there is no policy authorizing $t^\star$. Thus, similarly to the proof of Theorem 4.5.1, the **if** condition in Line 7 is satisfied. However, since P.2 does not apply, it means that: (i) there exists no data dependency that implies attribute $a^\star$; (ii) there exists no data dependency that implies attribute $a^\star$ such that the dependency determinant is explicitly authorized for u.

Let us consider the first case (i). Since no data dependency is available, the **if** condition in Line 11 is not satisfied, thus Algorithm jumps to Line 32, stops and returns an unauthorized message. As such a contradiction arises, since the assumption that $t^\star \in Rs(q^{rw})$ implies that a rewritten query is returned.
In the second case (ii), we have that there exists at least one data dependency $dd^\star$ with $a^\star \in dd^\star.Dep$ but there is no access control policy authorizing u to access $dd^\star.Det$. In this case, the **if** condition in Line 11 is satisfied (i.e., $dependenciesForAttribute \neq \emptyset$), but the $PoliciesForDeterminant$ set is empty. This brings the algorithm to remove $dd^\star$ from $dependenciesForAttribute$, thus making the **if** condition in Line 18 false. As such, algorithm jumps to line 29, stops and returns an unauthorized message. Thus, a contradiction arises, since the assumption that $t^\star \in Rs(q^{rw})$ implies that a rewritten query is returned. $\square$

Bibliography

[1] Oasis XACML 3.0 Standard. *http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf*.

[2] The Diaspora* Blog. *https://blog.diasporafoundation.org/*.

[3] The OneSocialWeb Project Website. *http://onesocialweb.org/*.

[4] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

[5] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: Anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 181–190, New York, NY, USA, 2007. ACM.

[6] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 135–146, New York, NY, USA, 2009. ACM.

[7] Filipe Beato, Markulf Kohlweiss, and Karel Wouters. Scramble! your social network data. In *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*, PETS'11, pages 211–225, Berlin, Heidelberg, 2011. Springer-Verlag.

[8] Gabriel Bender, Lucja Kot, and Johannes Gehrke. Explainable security for relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1411–1422, New York, NY, USA, 2014. ACM.

[9] Elisa Bertino. *Data protection from insider threats*, volume 4. Morgan & Claypool Publishers, 2012.

[10] Smriti Bhagat, Graham Cormode, Balachander Krishnamurthy, and Divesh Srivastava. Class-based graph anonymization for social network data. *Proc. VLDB Endow.*, 2(1):766–777, August 2009.

[11] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and Honggang Zhang. The growth of diaspora - a decentralized online social network in the wild. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 13–18, March 2012.

[12] Joachim Biskup, David W. Embley, and Jan-Hendrik Lochner. Reducing inference control to access control for normalized database schemas. *Inf. Process. Lett.*, 106(1):8–12, March 2008.

[13] Joachim Biskup, Sven Hartmann, Sebastian Link, and Jan-Hendrik Lochner. Efficient inference control for open relational queries. In *Proceedings of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy*, DBSec'10, pages 162–176, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Proceedings of the 11th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'05, pages 236–252, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] Alexander Brodsky, Csilla Farkas, and Sushil Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Trans. on Knowl. and Data Eng.*, 12(6):900–919, November 2000.

[16] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: Early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, NY, USA, 2009. ACM.

[17] A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD'08), in Conjunction with KDD'08*, 2008.

[18] B. Carminati, E. Ferrari, and J. Girardi. Trust and share: Trusted information sharing in online social networks. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1281–1284, April 2012.

[19] Barbara Carminati and Elena Ferrari. Access control and privacy in web-based social networks. *International Journal of Web Information Systems*, 4(4):395–415, 11 2008.

[20] Barbara Carminati, Elena Ferrari, and Andrea Perego. Rule-based access control for social networks. In *Proceedings of the 2006 International Conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET - Volume Part II*, OTM'06, pages 1734–1744, Berlin, Heidelberg, 2006. Springer-Verlag.

[21] Barbara Carminati, Elena Ferrari, and Andrea Perego. Enforcing access control in web-based social networks. *ACM Trans. Inf. Syst. Secur.*, 13(1):6:1–6:38, November 2009.

[22] Yuan Cheng, Khalid Bijon, and Ravi Sandhu. Extended rebac administrative models with cascading revocation and provenance support. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, SACMAT '16, pages 161–170, New York, NY, USA, 2016. ACM.

[23] L. A. Cutillo, R. Molva, and M. Önen. Safebook: A distributed privacy preserving online social network. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–3, June 2011.

[24] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, Dec 2009.

[25] Eman Yasser Daraghmi and Yuan Shyan Ming. Using graph theory to re-verify the small world theory in an online social network word. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications &#38; Services*, IIWAS '12, pages 407–410, New York, NY, USA, 2012. ACM.

[26] Harry S. Delugach and Thomas H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Trans. on Knowl. and Data Eng.*, 8(1):56–66, February 1996.

[27] D. E. Denning. Commutative filters for reducing inference threats in multilevel database systems. In *1985 IEEE Symposium on Security and Privacy*, pages 134–134, April 1985.

[28] Dorothy E. Denning. Annual review of computer science: Vol. 3, 1988. chapter Database Security, pages 1–22. Annual Reviews Inc., Palo Alto, CA, USA, 1988.

[29] M. Dürr, M. Maier, and F. Dorfmeister. Vegas – a secure and privacy-preserving peer-to-peer online social network. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pages 868–874, Sept 2012.

[30] W. F. Ehrsam, C. H. W. Meyer, J. L. Smith, and Tuchman W. L. Message verification and transmission error detection by block chaining. *US Patent 4074066*, 1976.

[31] Dóra Erdős, Rainer Gemulla, and Evimaria Terzi. Reconstructing graphs from neighborhood data. *ACM Trans. Knowl. Discov. Data*, 8(4):23:1–23:22, August 2014.

[32] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 351–360, New York, NY, USA, 2010. ACM.

[33] Csilla Farkas and Sushil Jajodia. The inference problem: A survey. *SIGKDD Explor. Newsl.*, 4(2):6–11, December 2002.

[34] Philip W.L. Fong. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, pages 191–202, New York, NY, USA, 2011. ACM.

[35] Charles Fried. Privacy. *The Yale Law Journal*, 77(3):475–493, 1968.

[36] Joachim von zur Gathen and Jrgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 3rd edition, 2013.

[37] Jennifer Golbeck. Using trust and provenance for content filtering on the semantic web. In *Proceedings of the Workshop on Models of Trust on the Web, at the 15th World Wide Web conference*, 2006.

[38] Liang Gou, Fang You, Jun Guo, Luqi Wu, and Xiaolong (Luke) Zhang. Sfviz: Interest-based friends exploration and recommendation in social networks. In *Proceedings of the 2011 Visual Information Communication - International Symposium*, VINCI '11, pages 15:1–15:10, New York, NY, USA, 2011. ACM.

[39] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 71–80, New York, NY, USA, 2005. ACM.

[40] John Hale and Sujeet Shenoi. Catalytic inference analysis: Detecting inference threats due to knowledge discovery. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 188–, Washington, DC, USA, 1997. IEEE Computer Society.

[41] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM.

[42] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. *Proc. VLDB Endow.*, 1(1):102–114, August 2008.

[43] Michael Hay, Gerome Miklau, David Jensen, Philipp Weis, and Siddharth Srivastava. Anonymizing social networks. *Technical report*, 2007.

[44] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia. Decent: A decentralized architecture for enforcing privacy in online social networks. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 326–332, March 2012.

[45] B. Kaliski. RFC 2315. *http://tools.ietf.org/html/rfc2315*, 1998.

[46] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM.

[47] Ralph Levien. PhD. Thesis: Attack Resistant Trust Metrics. *http://www.levien.com/thesis/compact.pdf*, 2004.

[48] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 93–106, New York, NY, USA, 2008. ACM.

[49] Wanying Luo, Qi Xie, and Urs Hengartner. Facecloak: An architecture for user privacy on social networking sites. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 03*, CSE '09, pages 26–33, Washington, DC, USA, 2009. IEEE Computer Society.

[50] M. Madejski, M. Johnson, and S. M. Bellovin. A study of privacy settings errors in an online social network. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 340–345, March 2012.

[51] Matthew Morgenstern. Controlling logical inference in multilevel database systems. In *Proceedings of the 1988 IEEE Conference on Security and Privacy*, SP'88, pages 245–255, Washington, DC, USA, 1988. IEEE Computer Society.

[52] Toshikazu Munemasa and Mizuho Iwaihara. Trend analysis and recommendation of users' privacy settings on social networking services. In *Proceedings of the Third International Conference on Social Informatics*, SocInfo'11, pages 184–197, Berlin, Heidelberg, 2011. Springer-Verlag.

[53] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 245–254, New York, NY, USA, 1999. ACM.

[54] National Institute of Standards and Technology. Announcing the Advanced Encryption Standard (AES). *http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf*, 2001.

[55] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.

[56] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[57] Xiaolei Qian, Mark E. Stickel, Peter D. Karp, Teresa F. Lunt, and Thomas D. Carvey. Detection and elimination of inference channels in multilevel relational database systems. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, SP '93, pages 196–, Washington, DC, USA, 1993. IEEE Computer Society.

[58] Delphine Reinhardt, Franziska Engelmann, and Matthias Hollick. Can i help you setting your privacy? a survey-based exploration of users' attitudes towards privacy suggestions. In *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia*, MoMM 2015, pages 347–356, New York, NY, USA, 2015. ACM.

[59] Christopher Riederer, Vijay Erramilli, Augustin Chaintreau, Balachander Krishnamurthy, and Pablo Rodriguez. For sale : Your data: By : You. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 13:1–13:6, New York, NY, USA, 2011. ACM.

[60] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 551–562, New York, NY, USA, 2004. ACM.

[61] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky. Vis-á-vis: Privacy-preserving online social networking via virtual individual servers.

In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–10, Jan 2011.

[62] Mohamed Shehab and Hakim Touati. Semi-supervised policy recommendation for online social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ASONAM '12, pages 360–367, Washington, DC, USA, 2012. IEEE Computer Society.

[63] Jiliang Tang, Charu Aggarwal, and Huan Liu. Recommendations in signed social networks. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 31–40, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[64] B. M. Thuraisingham. Security issues for data warehousing and data mining. In *Proceedings of the Tenth Annual IFIP TC11/WG11.3 International Conference on Database Security: Volume X : Status and Prospects: Status and Prospects*, pages 11–20, London, UK, UK, 1997. Chapman & Hall, Ltd.

[65] M. B. Thuraisingham. Security checking in relational database management systems augmented with inference engines. *Comput. Secur.*, 6(6):479–492, December 1987.

[66] Tyrone S. Toland, Csilla Farkas, and Caroline M. Eastman. The inference problem: Maintaining maximal availability in the presence of database updates. *Comput. Secur.*, 29(1):88–103, February 2010.

[67] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: Better privacy for social networks. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 169–180, New York, NY, USA, 2009. ACM.

[68] Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 555–566. VLDB Endowment, 2007.

[69] J. H. Wilkinson. The evaluation of the zeros of ill-conditioned polynomials. part i. *Numer. Math.*, 1(1):150–166, December 1959.

[70] Jierui Xie, Bart Piet Knijnenburg, and Hongxia Jin. Location sharing privacy preference: Analysis and personalized recommendation. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, IUI '14, pages 189–198, New York, NY, USA, 2014. ACM.

[71] Ching-man Au Yeung, Ilaria Liccardi, Kanghao Lu, Oshani Seneviratne, and Tim Berners-Lee. Decentralization: The future of online social networking. In *W3C Workshop on the Future of Social Networking Position Papers*, volume 2, pages 2–7, 2009.

[72] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Trans. on Knowl. and Data Eng.*, 12(3):372–390, May 2000.

[73] Tong Zhao, Chunping Li, Mengya Li, Qiang Ding, and Li Li. Social recommendation incorporating topic mining and social trust analysis. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 1643–1648, New York, NY, USA, 2013. ACM.