Supporting User Privacy Preferences in Digital Interactions

Sara Foresti and Pierangela Samarati

Dipartimento di Informatica
Università degli Studi di Milano
via Bramante 65, 26013
Crema (CR), Italy
email: sara.foresti@unimi.it, pierangela.samarati@unimi.it

ABSTRACT

The quick development and widespread adoption of Internet technologies allows servers to make available their services and resources to possibly unknown users anywhere any-time. To regulate access to such services in open scenarios, servers require users to release information about them through the disclosure of digital certificates. Since digital certificates, as well as access control policies may include sensitive information, it is necessary to define mechanisms that permit both the client and the server to specify privacy preferences to be considered in credential and policy disclosure.

In this chapter, we describe solutions supporting both client privacy preferences, and server disclosure policies. We illustrate the desiderata that these solutions should satisfy, and describe recent approaches that take client privacy preferences and server confidentiality into account in a negotiation process. Finally, we introduce some open issues that need further investigation.

1. INTRODUCTION

The advancements in ICT (Information and Communications Technology) allow users to take more and more advantage of the availability of online services (and resources) that can be accessed anywhere any-time. In such a scenario, the server providing the service and the requesting user may be unknown to each other. As a consequence, traditional access control systems [28] based on the preliminary identification and authentication of users requesting access to a service cannot be adopted, and are usually not suited to open scenarios (e.g., [13],[20],[29]). The solutions proposed to allow servers to regulate access to the services they offer, while not requiring users to manage a huge number of accounts, rely on *attribute-based access control* mechanisms (e.g., [7],[9],[16],[17],[18],[21],[23],[26],[34]). Policies regulating access to services define conditions that the requesting client must satisfy to gain access to the service of interest. Upon receiving a request to access a service, the server will not return a yes/no reply but it will send to the client the conditions that she must satisfy to be authorized to access the service. To prove to the server the possession of the attributes required to gain the access, the client releases *digital certificates* (i.e., *credentials*) signed by a trusted third party, the *certification authority*, who declares under its responsibility that the certificate holder possesses the attributes stated in the certificate. Practically, credentials are the digital representation of paper certificates (e.g., id card, passport, credit card). The adoption of credentials in access control has several advantages. First, credential-based access control enables clients to conveniently access Web services, without the need to remember a different <*username*, *password*> pair for each system with which she wants to interact. Second, it offers better protection against adversaries interested in improperly acquiring users access privileges.

The use of credentials to enforce access control restrictions in open environments has been widely studied in the last fifteen years. Most attention has however been devoted to the server-side of the problem, proposing a number of novel *policy languages*, for specifying access control rules (e.g., [7],[9],[23],[26],[34]); *policy engines*, for the evaluation of access requests and the enforcement of policy restrictions (e.g., [21],[23],[24],[34]); and strategies for *communicating* access conditions to the requesting clients, possibly engaging a negotiation protocol (e.g., [1],[21],[23],[24],[30],[31],[33],[34]). Since the interacting parties are assumed to be unknown to each other, the client may not known which attributes/credentials to release to gain access to the service of interest. As a consequence, the server should send to the client its policy, which may however be considered sensitive and therefore needs to be adequately protected before being disclosed. Most of the current

approaches implicitly assume that clients adopt an approach symmetric to the one used by servers for regulating access to the sensitive information certified by their credentials. Although expressive and powerful, these solutions do not fully support the specific protection requirements of the clients. In fact, clients are interested in a solution that is expressive and flexible enough to support an intuitive and user-friendly definition of the sensitivity/privacy levels that they perceive as characterizing their data. These preferences are used to choose which credentials to release when more than one subset of credentials satisfy the access control policy defined by the server (e.g., to buy a medicine, a patient needs to prove her identity by releasing either her identity card or her passport).

This chapter provides an overview of the privacy issues arising in open environments, both from the client and from the server points of view, and illustrates some solutions proposed to overcome these problems. The remainder of this chapter is organized as follows. Section 2 introduces basic concepts and describes the desiderata of privacy-aware access control systems operating in open environments. Section 3, Section 4, Section 5, and Section 6 illustrate some recent proposals that permit clients to specify privacy preferences that are then used to determine which credentials to disclose to gain access to a service of interest. Section 7 focuses on the server side of the problem, describing approaches that permit to regulate the disclosure of sensitive access control policies. Section 8 presents some open issues that still need to be addressed. Finally, Section 9 presents our concluding remarks.

2. BASIC CONCEPTS AND DESIDERATA

In this section, we describe the concepts at the basis of the proposals that we will describe in the following, and we discuss the desiderata that an attribute-based access control system should satisfy to effectively support both client and server privacy preferences.

2.1. Client Portfolio

The information that a client can provide to a server to gain access to a service are organized in a *portfolio* including both *credentials* signed by third parties and certifying client properties, and *declarations* stating uncertified properties uttered by the client [9]. Each credential c in the client portfolio is characterized by: a unique identifier $id(c)$, an issuer $issuer(c)$, a set of attributes $attributes(c)$, and a credential type $type(c)$. The type of a credential determines the set of attributes it certifies. Credential types are traditionally organized in a rooted *hierarchy*, where intermediate nodes represent abstractions defined over specific credential types that

correspond to the leaves of the hierarchy [3]. Formally, a hierarchy H of credential types is a pair $(T, \leq_{isa})$, where T is the set of all credential types and abstractions defined over them, and $\leq_{isa}$ is a partial order relationship on T. Given two credential types t_i and t_j, $t_i \leq_{isa} t_j$ if t_j is an abstraction of t_i. For instance, *photo_id* is an abstraction of credential types *id_card* and *passport* (i.e., $id\_card \leq_{isa} photo\_id$ and $passport \leq_{isa} photo\_id$). The root of the hierarchy is node *, representing any credential type. We note that declarations are usually modeled as a type of credentials, signed by the client herself. Figure 1 illustrates an example of a hierarchy of credential types.

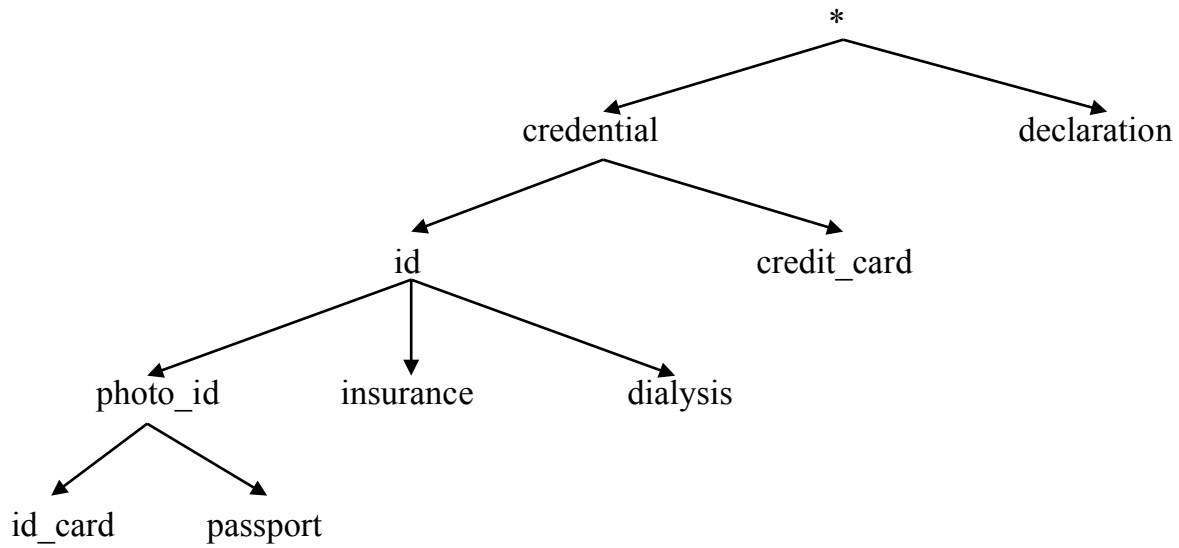

Figure 1 An example of hierarchy of credential types

The hierarchy of credential types is a knowledge shared between the client and the server. In fact, while a client knows exactly the different instances of credential types composing her portfolio, the server formulates its requests over credential types since it cannot be aware of the instances composing the client portfolio. We note that a client may possess different credentials of the same type (e.g., she can have more than one credit card).

Depending on the cryptographic protocol used for their generation, credentials can be classified as *atomic* or *non-atomic*. Atomic credentials are the most common kind of credentials used today in distributed systems (e.g., X.509 certificates) and can only be released as a whole. As a consequence, even if an atomic credential certifies attributes that are not required to gain access to a service, if the client decides to release it, these attributes will be disclosed to the server. Non-atomic credentials have recently been proposed as a successful approach to limit data disclosure (e.g., U-Prove and Idemix [10],[11]). Credentials generated adopting these technologies permit the client

to selectively release a subset of the attributes certified by the credential (as well as the existence of the credential itself). Note that the release of an attribute (or a set thereof) certified by a non-atomic credential entails the disclosure of the existence in the client portfolio of the credential itself. Clearly, declarations are non-atomic credentials.

Attributes within credentials are characterized by a *type*, a *name*, and a *value* (e.g., attribute *Name* of type *Name* with value *Bob*), which can either depend only on the client or on the specific credential certifying the attribute. In the first case, the attribute is *credential-independent* since its value is the same, independently from the credential certifying it (e.g., *Name* and *DoB* are credential-independent attributes). In the second case, the attribute is *credential-dependent* since its value depends not only on the credential holder, but also on the specific instance of the credential certifying it (e.g., attribute type *CCNum*, representing the credit card number, is a credential-dependent attribute since each credit card has a different number).

$id(c)$	Atomic	$type(c)$	$attributes(c)$
MyIdCard	✓	id_card	Name, DoB, City
MyPassport		passport	Name, DoB, Country
MyVISA	✓	credit_card	Name, *VISANum*, *VISALimit*
MyAmEx	✓	credit_card	Name, *AmExNum*, *AmExLimit*
MyDialysis		dialysis	Name, City
MyInsurance	✓	insurance	Name, *Company*, *Coverage*
MyDecl		declaration	Name, DoB, City, Country, *VISANum*, *VISALimit*, *AmExNum*, *AmExLimit*, *Company*, *Coverage*, e-mail

Table 1 An example of client portfolio

For instance, Table 1 illustrates an example of client portfolio composed of four atomic and three non-atomic credentials. In the figure, credential-independent attributes are in roman, while credential-dependent attributes are in *italic*.

2.2. Disclosure Policies

Attribute-based access control restricts access to server service depending on the attributes and credentials that the requesting client discloses to the server. The policy regulating access to services is therefore defined over attributes and credentials provided by clients. Since the server may not know the requesting client and therefore ignore the credential instances in her portfolio, the access control policy is defined considering only the hierarchy of credential types, which represents a common knowledge to the interacting parties. An access control policy is defined as a Boolean formula composed of basic conditions $cond$ of the form $(term_1 o\ term_2)$, where o is

a predicate operator (e.g., >, <, =), and $term_1$ and $term_2$ are its operands. The operands of a basic condition can be either constant values, or (certified or declared) attributes represented by terms of the form $c.a$, where c is a variable representing a credential and a is the name of the attribute. For instance, basic condition *Coverage*>10,000 USD requires that the coverage offered by insurance is higher than 10,000 USD to access the service. The server may also define restrictions on the type of credentials that should certify the requested attributes and/or require that a set of attributes in the policy are certified by the same credential. As an example, policy ($type(c)$=*insurance*) ∧ (c.*Company*≠'A') ∧ (c.*Coverage*>10,000 USD) requires that attributes *Company* and *Coverage* are certified by the same credential c, of type *insurance*.

2.3. Trust Negotiation

Since clients and servers operating in open environments are assumed to be unknown to each other, they interact to the aim of building a trust relationship that permits the client to gain access to a service offered by the server. This trust relationship is built step by step through the exchange of credentials. Since credentials may certify sensitive information, their release is often regulated, like for services, by access control policies. Usually, these conditions require the release by the counterpart of another credential (or set thereof). As an example, a user agrees to release the certificate stating her dialysis condition to a server only if the server proves (through a certificate) to be a medical institution or a pharmacy recognized by the Health Ministry.

To gain access to a service, the client and the server must then find a sequence of certificates exchange, called *strategy*, satisfying the access control policies of both parties. For instance, with reference to the above example, a successful strategy that permits the user to buy the medicine of interest consists of the following steps: *i)* the patient sends her request to the pharmacy; *ii)* the pharmacy answers with a request for a certificate proving that the user has a nephrological disease; *iii)* the client, in turn, asks the pharmacy the certificate proving that it is recognized by the Health Ministry; *iv)* the pharmacy releases to the client the requested certificate; *v)* the client then discloses to the pharmacy her dialysis certificate; *vi)* finally, the server grants access to the service. Different approaches have been proposed in the literature to the aim of identifying a successful trust negotiation strategy (e.g., [1],[21],[23],[24],[30],[31],[33],[34]) that depends not only on the policies defined by the parties and on the credentials at their disposal, but also on their choice of disclosure/non-disclosure of their data. As an example, an eager strategy would

disclose a credential as soon as the policy regulating its release is satisfied, while a more parsimonious strategy permits the release of a credential only if there exists a *successful strategy* that will finally grant the client access to the service. It is interesting to note that, given the policies and credentials of the interacting client and server, there may exist more than one successful strategy. For instance, with reference to the portfolio in Table 1, policy $(type(c)=id) \wedge (c.DoB<01/01/1994)$ can be satisfied by the client releasing either credential *MyIdCard* or credential *MyPassport*. Although all the successful strategies may seem equivalent, this is generally not true. Both the client and the server may prefer to release a credential over another one because they perceive a different sensitivity level associated with the information that credentials certify. For instance, the client may prefer to release her *id_card* over her *passport*.

2.4. Client Privacy Preferences

Given the server request, the client needs to determine which credentials and/or attributes to disclose to satisfy it. This task becomes harder if different subsets of credentials and/or attributes in the client portfolio can be used to fulfill the server policy, since the client needs to choose among them. Ideally, the choice should be driven by the sensitivity level that the client perceives for her credentials and attributes, as she will be more willing to disclose less sensitive portfolio components. It is therefore necessary to provide clients with a flexible and effective system that automatically determines the release strategy that better satisfies her privacy preferences. To this purpose, a flexible and expressive model for representing privacy preferences needs to be defined. We now illustrate the main desiderata that a privacy-aware access control system should satisfy [5].

- *Fine-grained preference specification*. The privacy preferences associated with attributes and credentials in the client portfolio reflect the sensitivity perceived by the credential owner for the personal information represented by the attribute/credential. The model should support the definition of privacy preferences for each instance of attribute and credential in the client portfolio, meaning that different instances of the same credential type (and credential-dependent attribute) might be associated with different privacy preferences. For instance, with reference to the portfolio in Table 1, the client may prefer to release VISA credit card instead of AmEx.
- *Inheritance of privacy preferences*. To provide flexibility in the definition of privacy preferences and a user-friendly mechanism for their specification, the model should take advantage of the hierarchy of credential types characterizing the client portfolio. When the client portfolio is composed of a huge number of

attributes and credentials, it might be difficult for the client to specify a different preference value for each credential and attribute. Privacy preferences associated with abstractions of credential types could however be inherited by all its specifications, if not overwritten by a more specific preference value, thus reducing the client overhead. For instance, with reference to the hierarchy of credential types in Figure 1 and the portfolio in Table 1, the client may specify a single privacy preference associated with credential type *photo_id*, which is automatically inherited by credentials *MyIdCard* and *MyPassport*.

- *Partial order relationship and composition operator*. The domain of privacy preferences should be characterized by a (partial) order relationship $\succcurlyeq$ that permits to precisely determine whether a given piece of personal information is more or less sensitive than another. The domain should also be characterized by a composition operator $\oplus$, which permits to compute the privacy preference value characterizing the release of a set of attributes and/or credentials. As an example, if the domain of privacy preferences is the set of positive integer numbers, the partial order relationship could be the "greater than" relationship (i.e., $\geq$), while the composition operator could be the sum operator (i.e., +).
- *Sensitive associations*. In different scenarios, the combined release of a set of attributes and/or credentials is considered more (or less) sensitive than the release of each portfolio component singularly taken. For instance, with reference to the portfolio in Table 1, the client may consider the combined release of attributes *DoB* and *City* more sensitive than the release of each of the two attributes, since their combination could be exploited to infer the identity of the client [27]. On the other hand, she may value the release of *City* and *Country* less sensitive than the release of the two attributes singularly taken, due to the dependency between the values of the two attributes. As a consequence, the model should support the definition of a privacy preference value for the combined release of a set of attributes and/or credentials that is different from the result of the combination of the privacy preferences of the items in the set.
- *Disclosure constraints*. There are situations where the client needs to specify restrictions on the combined release of portfolio components, since she wants to keep the association among a subset of attributes and/or credentials confidential, or limit their combined release. For instance, with reference to the portfolio in Table 1, the client may not be willing to release credential *MyDialysis* together with attribute *DoB*, to prevent the server from exploiting this information for data mining purposes (e.g., to analyze the age of people with nephrologic diseases).

- *Context-based preferences*. The privacy preferences associated with attributes and credentials may vary depending on the context in which their release is requested (i.e., depending on the requested service and/or on the server providing it). For instance, the client may be more willing to release her dialysis certificate to a pharmacy for buying a medicine than to a hotel for booking a room.
- *History-based preferences*. The preference of the client toward disclosing one credential (attribute, respectively) over another one may depend on the history of past interactions with the server offering the service. As a matter of fact, if the server already knows the attributes and credentials released by the client during a previous interaction, the client may be more willing to release the same (or a different) set of portfolio components. For instance, with reference to the portfolio in Table 1, assume that the client released credential *MyVISA* to a server to buy a service. When interacting again with the same server to buy another service, the client may prefer to use the same credit card, instead of releasing also credential *MyAmEx*.
- *Proof of possession*. Thanks to novel technologies, clients can release proofs of possession of certificates and proofs of the satisfaction of conditions (e.g., [10],[11]). As a consequence, the model should also permit the client to specify privacy preferences associated with proofs (besides attributes and credentials on which proofs are defined). For instance, with reference to the portfolio in Table 1, the client may consider more sensitive the release of her *DoB* than the release of a proof that she is at least 18.
- *User-friendly preference specification*. The definition of privacy preferences should be easy for the client, who may not be familiar with access control systems. As a consequence, it is necessary to provide clients with interfaces that permit to easily define preferences without introducing inconsistencies.

2.5. Server Privacy Preferences

With attribute-based access control, servers regulate access to their services based on the attributes and certificates presented by the requesting client. Upon receiving an access request, the server needs to communicate to the client the policy that she should satisfy to possibly gain access to the service. The access control policy could however be sensitive and the server may not be willing to disclose it completely to the client: while the communication of the complete policy favors the privacy of the client (since she can avoid disclosing her attributes and credentials if they would not satisfy the conditions in the policy), the communication of the attributes involved in the policy only favors the privacy of the server (since the specific conditions are not disclosed). Also, different portions of the same policy may be subject to different confidentiality

requirements. For instance, assume that a pharmacy grants to clients access to the online medicine purchase service only if the insurance coverage of the clients is higher that 10,000 USD and the insurance company is not in the pharmacy black list. The pharmacy might not mind disclosing the fact that only clients with insurance cover greater than 10,000 USD can access its services, but it does not want to reveal its black list.

The system managing the disclosure of server policies should satisfy the following desiderata.

- *Disclosure policy*. The server should be able to define, at a fine-granularity level, how policy release should be regulated.
- *Policy communication*. The communication of the access control policy regulating access to the requested service to the client should guarantee that privacy requirements are satisfied and that the client has enough information to determine the set of attributes and/or credentials she needs to disclose to possibly gain access to the service. It is therefore necessary to define a mechanism that adequately transforms the access control policy before communicating it to the client.
- *Integration with client mechanisms*. The approach designed to regulate policy release should be integrated with the one designed to manage the release of portfolio components at the client side.

Note that in a negotiation process, both the client requesting access to a service and the server providing it possess a portfolio and regulate the disclosure of credentials and attributes composing it according to their access control policy.

3. COST-SENSITIVE TRUST NEGOTIATION

A solution that takes disclosure preferences into consideration in attribute-based access control has been introduced in [12]. The authors propose to associate a *sensitivity cost* $w(c)$ with each credential c in the client (and server) portfolio, and with each access control policy p regulating credentials disclosure and access to services. A policy p is defined as a Boolean formula over the credentials in the counterparty's portfolio. Boolean variable representing credential c in policy p is *true* if c has already been disclosed, it is *false* otherwise. The sensitivity cost associated with credential c (policy p, respectively) models how much the credential's owner (party who defined the policy, respectively) values the release of the credential (policy, respectively) and the disclosure of the sensitive information that the credential certifies. Intuitively, a client (server, respectively) is more willing to disclose credentials (policies, respectively) with lower sensitivity cost and, vice versa, she prefers to keep credentials (policies,

respectively) with high sensitivity cost confidential. For instance, Table 2 (Table 3, respectively) illustrates an example of client portfolio (server portfolio, respectively) . For each credential, the table reports the policy regulating its disclosure, the sensitivity cost of the credential, and the sensitivity cost of it policy. Constant value TRUE is used in policy definition to model the case when the release of a credential is free, that is, it is not regulated by a policy. (The portfolio in Table 2 is a simplified version of the portfolio in Table 1.)

$id(c)$	$w(c)$	Policy p regulating c	$w(p)$
MyIdCard	2	TRUE	0
MyPassport	4	TRUE	0
MyCreditCard	10	POS_register	5
MyDialysis	20	pharmacy_register	10
MyInsurance	15	pharmacy_register	10

Table 2 An example of client portfolio and policies regulating its disclosure

$id(c)$	$w(c)$	Policy p regulating c	$w(p)$
MyPOSRegister	2	TRUE	0
MyPharmacyRegister	5	passport ∨ id_card	4

Table 3 An example of server portfolio and policies regulating its disclosure

The goal of the client and the server engaging a negotiation protocol is that of *minimizing* the sensitivity cost of the credentials and policies exchanged during a successful negotiation strategy. This optimization problem can be formulated as follows [12].

Problem 1: **Minimum Sensitivity Cost problem -** Let C_s be the set of server credentials and services; P_s be the set of policies regulating the disclosure of server credentials and access to services; C_c be the set of client credentials; P_c be the set of policies regulating the disclosure of client credentials; w: $C_s \cup P_s \cup C_c \cup P_c \rightarrow \mathbb{R}$ be the sensitivity cost function; and $s \in C_s$ be the service requested by the client. Find an exchange sequence of credentials and policies such that:

1. s is released to the client;
2. the policy regulating the disclosure of each credential released to the counterpart is satisfied before credential release;
3. the sum of the sensitivity costs of released credentials and policies is minimum.

The problem of computing a Minimum Sensitivity Cost strategy is NP-hard [12] and therefore any algorithm that solves it at optimum has exponential cost in the size of its input (i.e., the number of credentials and policies in $C_s \cup P_s \cup C_c \cup P_c$). In [12] the

authors propose two different heuristic approaches for computing a good (although non optimal) solution to the problem. These heuristics have polynomial computational complexity and can be adopted when policies can be freely disclosed, and when they are associated with a sensitivity cost, respectively.

Non-sensitive Policies. The solution proposed for the simplified scenario where policies are not associated with a sensitivity cost (i.e., they can be freely released) is based on the definition of a *policy graph* modeling the policies regulating credential disclosure at both the client and server side. A policy graph $\boldsymbol{G(V, A, w)}$ is defined as a weighted graph with:

- a vertex $\boldsymbol{v_c}$ for each credential $\boldsymbol{c}$ in $\boldsymbol{C_s} \cup \boldsymbol{C_c}$;
- a vertex v_s for each service s in C_s;
- a vertex v_T for constant value TRUE;
- a vertex v for each disjunction in the policies regulating credential release;
- an edge (v_i, v_j), with v_i and v_j vertexes representing credentials, if the release of the credential represented by v_i is a necessary condition to gain access to the credential represented by v_j;
- an edge (v_i, v_j), with v_i a vertex representing a credential and v_j a vertex representing a disjunction, if v_i is one of the clauses of the disjunction represented by v_j.

The weight of a vertex representing a credential corresponds to the sensitivity cost of the credential it represents, while other vertexes do not have weight. For instance, consider the access control policies in Table 2 and Table 3 and service *MedicineBooking*, regulated by policy p = *dialysis* ∨ (*id_card* ∧ (*credit_card* ∨ *insurance*)). Figure 2(a) illustrates the policy graph modeling the access control policies in the system.

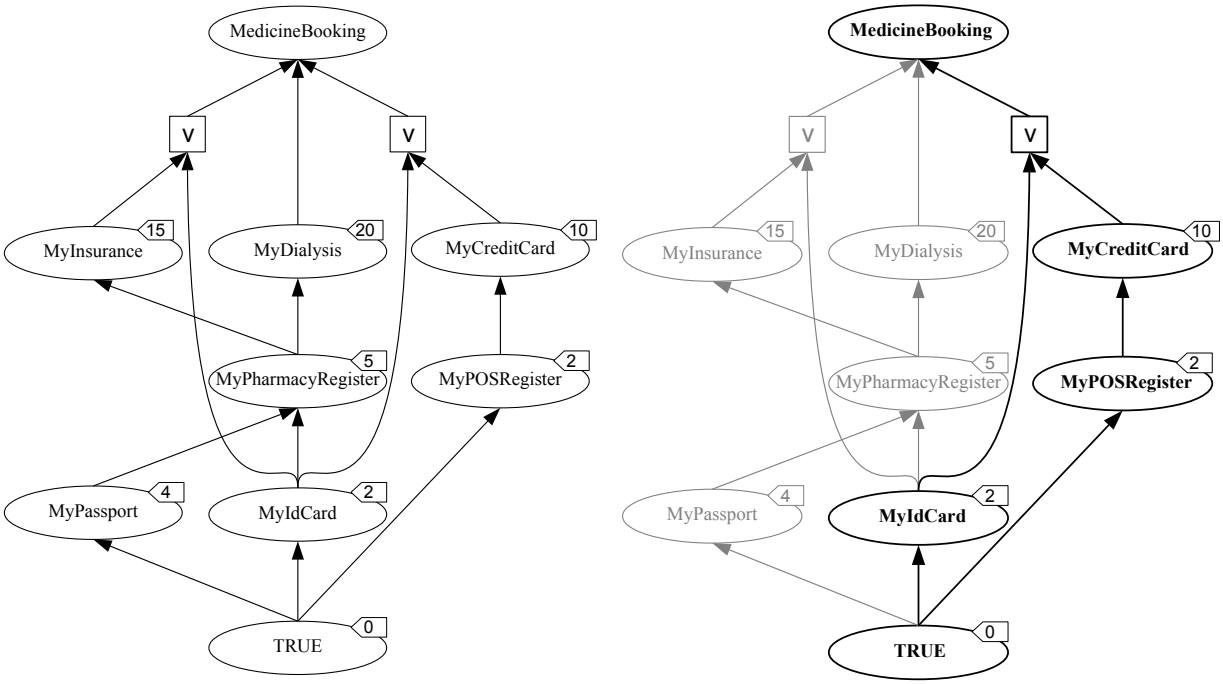

Figure 2 Policy graph for the policies in Table 2 and in Table 3 (a), and Minimum Directed Acyclic graph for the *MedicineBooking* service (b)

The first step of the negotiation process consists in disclosing the policies regulating credential release and access to services at the client and at the server side. This information permits to correctly build the policy graph. Note that this disclosure is permitted thanks to the assumption that policies are not sensitive in this simplified scenario. The minimum sensitivity cost problem then translates into the equivalent problem of determining a *Minimum Directed Acyclic Graph* for the policy graph, starting at vertex v_T (representing value TRUE) and ending at the vertex v_s representing the requested service s. Formally, a Minimum Directed Acyclic Graph is defined as follows.

Definition 1: Minimum Directed Acyclic Graph - Let $G(V, A, w)$ be a policy graph, v_T be the vertex representing value TRUE, and v_s be the vertex representing service s. A directed acyclic graph starting at v_T and ending at v_s is a sub-graph $G'(V', A', w)$ of G such that:

1. G' is acyclic;
2. $v_T, v_s \in V'$;
3. $\nexists(v_i, v_T) \in A', v_i \in V'$;
4. $\nexists(v_s, v_i) \in A', v_i \in V'$;
5. $\forall\, v_i \in V', \exists \langle v_T | v_i \rangle$, where $\langle v_T | v_i \rangle$ is a path starting at v_T and ending at v_i;

6. $\forall\, v_i \in V'$, $\forall (v_i, v_j) \in A$, where v_i represents a credential, $(v_i, v_j) \in A', v_j \in V'$;
7. $\nexists G''(V'', A'', w)$ that satisfies all the previous conditions and such that $\sum_{v \in V''} w(v) < \sum_{v \in V'} w(v)$.

It is easy to see that a directed acyclic graph starting at v_T and ending at v_s represents a successful negotiation strategy for service s. Therefore, the minimum sensitivity cost problem and the problem of computing a minimum directed acyclic graph from vertex v_T to vertex v_s are equivalent. The heuristic algorithm proposed in [12] is based on a variation of the well-known Dijkstra algorithm [14]. In [12] the authors experimentally prove that the proposed algorithm computes an optimal solution in most cases. For instance, consider the policy graph in Figure 2(a) and assume that the client is interested in the *MedicineBooking* service. Figure 2(b) illustrates a Minimum Directed Acyclic Graph for the *MedicineBooking* service with cost 14, where the vertexes and edges in the policy graph that also belong to the Minimum Directed Acyclic Graph are in black, while the other vertexes and edge are in gray.

Sensitive Policies. The solution proposed in [12] for the more complex scenario where both credentials and policies regulating their release are associated with a sensitivity cost is based on a *greedy strategy* that consists of two steps. During the first step, the interacting parties adopt an eager strategy (i.e., each party discloses to the counterpart the name of a credential as soon as the policy for its release is satisfied) to mutually exchange the name and sensitivity cost associated with credentials that could be useful for identifying a successful negotiation strategy with minimum cost. If this first step finds such a strategy, the client and the server start the second step of the protocol, which consists in enforcing the strategy discovered during the first step. For instance, with reference to the policy graph in Figure 2(a), the first step consists of the sequence of releases illustrated in Figure 3. First, the client and the server reveals to each other the name and sensitivity cost of credentials whose release is not regulated by a policy, that is, *MyIdCard* and *MyPassport* for the client and *MyPOSRegister* for the server. These releases satisfy the policy regulating the release of *MyCreditCard* at the client side and *MyPharmacyRegister* at the server side, whose names and sensitivity costs are disclosed. These releases, in turn, satisfy the policies regulating the disclosure of credentials *MyInsurance* and *MyDialysis* at the client side and service *MedicineBooking* at the server side. The exchange then represents a successful negotiation strategy. Note that the edges in Figure 3 are labeled with the cumulative sensitivity cost of the negotiation process (e.g., *MyCreditCard* is associated with cost 12 = w(*MyCreditCard*) + w(*MyPOSRegister*)). The successful negotiation strategy

computed by the first step is enforced during the second step of the protocol. Therefore, the server first discloses credential *MyPOSRegister* and the client releases *MyIdCard*. When the client receives the credential from the server, she discloses *MyCreditCard*, thus gaining access to the *MedicineBooking* service. The overall sensitivity cost of the strategy is 14.

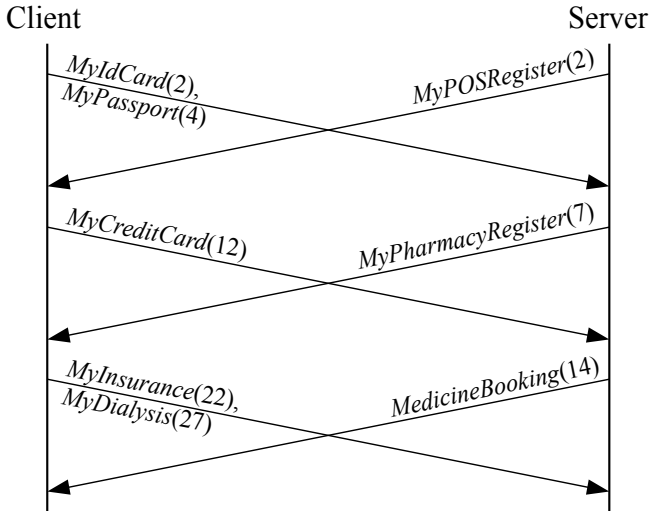

Figure 3 Sequence of exchanges between the client and the server to determine a successful negotiation strategy

Open Issues. It is interesting to note that, although effective, the model and algorithms proposed in [12] suffer from some limitations. A first drawback is that the proposed approach assumes that the disclosure of access control policies does not need to be regulated, while also policy release may be subject to restrictions. Also, this solution assumes that the objective of a privacy-aware negotiation protocol is that of minimizing the overall sensitivity cost of credentials and policies disclosed during the negotiation process. However, the goal of the two parties may be different. For instance, with reference to our example, the pharmacy offering the *MedicineBooking* service may not be interested in minimizing the sensitivity cost of the policies and credentials it needs to disclose to offer the service. On the contrary, the patient wants to minimize the sensitivity cost of the credentials she must disclose to the pharmacy. We also note that the model in [12] does not satisfy all the desiderata illustrated in Section 2 to support privacy preferences in attribute-based access control scenarios. In fact, it only supports the definition of privacy preferences as sensitivity costs, which have a numerical domain characterized by a total order relationship (i.e., $\geq$) and by a composition operator (i.e., +).

4. POINT-BASED TRUST MANAGEMENT

The problem of minimizing the amount of sensitive information disclosed by a trust negotiation protocol has been also addressed in [32], where the authors propose a *point-based trust management model*. This model assumes that policies regulating access to services and release of credentials are based on the definition of quantitative measures. More precisely, the server associates a number pt of *points* with each credential type t. This value represents the trustworthiness perceived by the server for the credential issuer (i.e., credentials issued by a more reliable party will be associated with a higher number of points and vice versa). To restrict the access to its services, the server then associates a *threshold* thr with each service. To gain access to a service s, the client must disclose a subset of credentials in her portfolio such that the sum of the points of the released credentials is higher than or equal to the threshold fixed by the server for s. Analogously, the client associates a *privacy score* ps with each credential in her portfolio, which represents how much she values the release of the credential to an external server. The higher the privacy value of a credential, the lower the client willingness in its release. As a consequence, a client who is interested in accessing a service s must determine a subset of credentials in her portfolio that satisfies the threshold fixed by the server for s, while minimizing the privacy score of released credentials. Table 4 illustrates an example of points and privacy scores associated by the server and the client, respectively, to the credentials composing the client's portfolio.

	id_card	passport	credit_card	dialysis	insurance
pt	1	1	2	3	2
ps	2	4	10	20	15

Table 4 An example of points pt and privacy scores ps associated by the server and the client, respectively, to the credential in the client portfolio

Since the server policy might be considered sensitive, the server does not reveal the threshold associated with its services to the client. Analogously, the client does not reveal to the counterpart the privacy scores she associates with the credentials in her portfolio. As a consequence, when a client requests access to a service, she needs to identify a subset of the credentials in her portfolio that satisfies the server threshold (i.e., the access control policy regulating the release of the service) without knowing it and without revealing to the server credentials' privacy scores. More formally, the *Credential Selection* problem is an optimization problem that can be formulated as follows [32].

Problem 2: **Credential Selection problem -** Let $C = \{c_1, \dots, c_n\}$ be the set of credentials in the client portfolio; $pt(type(c_i))$ be the points associated by the server with credential type $type(c_i)$, $i = 1, \dots, n$; $ps(c_i)$ be the privacy score associated by the client with credential c_i, $i = 1, \dots, n$; s be the service requested by the client; and thr be the release threshold associated with s. Find a subset $D \subseteq C$ of credentials such that:

1. $\sum_{c \in D} pt(type(c)) \geq thr$;
2. $\nexists D' \subseteq C$ s.t. $\sum_{c \in D'} pt(type(c)) \geq thr$ and $\sum_{c \in D'} ps(c) < \sum_{c \in D} ps(c)$.

The first condition states that the subset of credentials in the client portfolio must satisfy the server policy, while the second condition states that the sensitive information disclosed is minimum. For instance, with reference to the points and privacy scores in Table 4, let us assume that the server offering service s (*MedicineBooking* in our example) defines a threshold thr=3. The release of her *id_card* and of her *credit_card* permits the client to gain access to the service of interest (pt(*id_card*) + pt(*credit_card*) = 3 $\geq thr$), while minimizing the overall privacy score of released information (ps(*id_card*) + ps(*credit_card*) = 12).

Dynamic Programming Algorithm. The Credential Selection problem is NP-hard and can be rewritten into a knapsack problem, where each credential c can be inserted into the knapsack with weight $pt(type(c))$ and value $ps(c)$ [32]. Since the knapsack algorithm maximized the value of the items inserted in the knapsack to satisfy its capacity, while the goal of the client is that of minimizing the sensitivity of the credentials necessary to reach the threshold of interest, the solution to the credential selection problem is computed by inserting in the knapsack those credentials that will not be released. Intuitively, the knapsack problem is complementary to our problem and therefore the approach in [32] finds the complementary solution to the Credential Selection problem by exploiting a known dynamic programming algorithm for the knapsack problem [14]. The knapsack capacity KC is computed as the complementary of the threshold fixed by the server with respect to the clients portfolio, that is, $KC = \sum_{c \in C} pt(type(c)) - thr$, which is the difference between the sum of points associated with credential types in the client's portfolio and the threshold fixed by the server to gain access to the service. With reference to the example above, $KC = (1 + 1 + 2 + 3 + 2) - 3 = 6$.

The dynamic programming solution to the knapsack problem is based on the definition of a matrix M with $n + 1$ rows, where n is the number of items that can be inserted into the knapsack (i.e., credentials in our scenario), and $KC + 1$ columns. All the cells

in the first row and in the first column of the matrix are set to zero (i.e., $M[i,0]=0$, $i=0,\dots,n$, and $M[0,j]=0$, $j=0,\dots,KC$). The value of the other cells in the matrix is computed according to the following formula:

$$M[i,j]=\begin{cases}M[i-1,j], & j<pt(type(c_i))\\ \max\left(M[i-1,j],M[i-1,i-pt(c_i)]+ps(c_i)\right), & j\geq pt(type(c_i))\end{cases}$$

The values of the cells in the matrix are computed, in the order, starting from top to bottom and from left to right (i.e., by increasing value of i and j, respectively). Each cell in the matrix represents the total value of the knapsack, obtained inserting (a subset) of the items preceding the current element in the matrix without exceeding the knapsack capacity. It is obtained as the current value of the knapsack either including or not including the current element.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
id_card	0	2	2	2	2	2	2
passport	0	4	6	6	6	6	6
credit_card	0	4	10	14	16	16	16
dialysis	0	4	14	20	24	30	34
insurance	0	4	15	20	29	35	39

Table 5 An example of dynamic programming matrix for the portfolio in Table 4

Table 5 illustrates the matrix computed considering points and privacy scores in Table 4. The first row in the matrix represents an empty knapsack. Cells M[*id_card*,1],…, M[*id_card*,6] in the first row model the insertion of credential *id_card* in the empty knapsack. As a consequence, the knapsack has weight 2. Cell M[*passport*,2] is obtained by comparing the solution represented by cell M[*id_card*,2] (which models a knapsack including only *id_card*) with the solution M[*id_card*,1]∪{*passport*} obtained by inserting also credential *passport* into the knapsack. The weight of the two alternative solutions is, respectively, 2 and 2+4=6. Since 6>2, M[*passport*,2] is set to 6 and it represents a knapsack including credentials *id_card* and *passport*. The other cells in the matrix are computed in the same way.

The optimal solution to the knapsack problem is represented by the value in cell $M[n,KC]$, which represents the value of the knapsack obtained trying to insert all the candidate elements in the knapsack without exceeding its capacity. To determine the elements that belong to the optimal solution, it is necessary to keep track of which item has been inserted at each step. For instance, consider the matrix in Table 5, cell M[*insurance*,6]=39 is as the sum of the cells in gray in the table, that is, it represent a

solution including credentials *passport*, *dialysis*, and *insurance*. Since the credentials included in the knapsack are not disclosed, the credentials disclosed by the client to gain the access are *id_card* and *credit_card* that, as already noted, satisfy the threshold fixed by the server for the *MedicineBooking* service while minimizing privacy scores.

The traditional dynamic programming algorithm described above for the knapsack problem assumes that the client knows the points assigned by the server to credential types (or that the server knows the privacy scores that the client associates with the credentials in her portfolio). Since this assumption does not hold in the considered scenario, in [32] the authors propose to enhance the basic algorithm to permit the client and the server to interact with each other for computing a solution to the knapsack problem without the need for the client and the server to reveal to each other their secret parameters. The proposed solution consists of a secure two-party dynamic-programming protocol, which relies on homomorphic encryption to provide privacy guarantees to sensitive information [15],[25].

Open Issues. The model and algorithm introduced in [32] suffer from different shortcomings. First of all, the client and the server must share, as a common knowledge, the set of possible credentials on which the negotiation process should be based. Such knowledge may however put the privacy of the server policy at risk. The proposed model also assumes that the access control policy defined by the server consists of a threshold value, but in many real-world scenarios the server needs to define more expressive policies. Furthermore, the focus of the proposal, as well as the model in [12], is more on the negotiation process than on the management of the privacy preferences of the interacting parties. The solution in [32] represents however an important step towards the definition of a privacy-aware access control model, even if it does not satisfy all the desiderata described in Section 2. In fact, this approach only supports the definition of privacy preferences as privacy scores, which have a numerical domain characterized by a total order relationship (i.e., $\geq$) and by a composition operator (i.e., +).

5. LOGICAL-BASED MINIMAL CREDENTIAL DISCLOSURE

The solutions in [12], [32] are based on the assumption that privacy preferences can be expressed as numerical values, defined over a domain characterized by a total order relationship and an additive operator. While this assumption permits to easily integrate privacy preferences with traditional negotiation processes, the usability of the resulting system may be limited. In fact, it might not be easy for the final user to express her privacy preferences through numeric values, also because the adoption of numeric preference values may cause unintended side effects (e.g., dominance relationships are

not explicitly defined, but are implied by the values assigned to the portfolio components). To overcome these limitations, in [22] the authors propose to adopt *qualitative* (instead of quantitative) preference values. The proposed solution is based on the assumption that credentials are singleton (i.e., certify one attribute only) and that the policy defined by the server is publicly available. The goal of the approach is to determine, among the successful negotiation strategies, the one that better suits the client preferences (i.e., the set of credentials that minimizes the amount of sensitive information disclosed to the server to gain access to the requested service). When the number of successful strategies is limited, the client can explicitly choose the one she prefers. However, when the number of credentials in the client portfolio increases and the server policy becomes complex, the number of successful trust negotiation strategies may grow quickly. For instance, assume that the client portfolio is composed of credentials {*Name*, *DoB*, *City*, *VISANum*, *VISALimit*, *AmExNum*, *AmExLimit*, *Insurance*, *InsCoverage*, *Dialysis*}, and that the policy regulating access to the *MedicineBooking* service is ((*Name* ∧ (*DoB* ∨ *City*) ∨ *Dialysis* ∨ *Insurance*) ∧ ((*VISANum* ∧ *VISALimit*) ∨ (*AmExNum* ∧ *AmExLimit*) ∨ (*InsCoverage* ∧ *DoB*)). There are 12 strategies that satisfy the access control policy. It is therefore necessary to define a mechanism that permits to exploit qualitative disclosure preferences defined by the client to limit the number of strategies among which she is explicitly asked to choose.

Qualitative Preferences. Given the set $C = \{c_1, \dots, c_n\}$ of credentials in the client portfolio, the release of a subset of credentials is modeled as a binary n-dimension vector D, where $D[i] = 1$ if c_i is released and $D[i] = 0$ otherwise, $i = 1, \dots n$. For instance, with reference to the previous example, Table 6 summarizes the subsets of portfolio credentials satisfying the policy regulating service *MedicineBooking*. Disclosure D_1 represents the release of {*Name*, *DoB*, *VISANum*, *VISALimit*}.

	Name	DoB	City	VISANum	VISALimit	AmExNum	AmExLimit	Insurance	InsCoverage	Dialysis
D_1	1	1	0	1	1	0	0	0	0	0
D_2	1	1	0	0	0	1	1	0	0	0
D_3	1	1	0	0	0	0	0	0	1	0
D_4	1	0	1	1	1	0	0	0	0	0
D_5	1	0	1	0	0	1	1	0	0	0
D_6	1	1	1	0	0	0	0	0	1	0
D_7	0	0	0	1	1	0	0	0	0	1
D_8	0	0	0	0	0	1	1	0	0	1
D_9	0	1	0	0	0	0	0	0	1	1
D_{10}	0	0	0	1	1	0	0	1	0	0
D_{11}	0	0	0	0	0	1	1	1	0	0
D_{12}	0	1	0	0	0	0	0	1	1	0

Table 6 Disclosure strategies that satisfy the access control policy of service *MedicineBooking*

The model proposed in [22] permits to specify privacy preferences at different granularity levels. Dominance relationship $\succ_i$ defines disclosure preferences for credential c_i. Usually, credential-level preferences state that $0 \succ_i 1,\ i = 1, \ldots n$, meaning that the client prefers not to disclose credential c_i. To compare the disclosure of different subsets of credentials in the client portfolio, credential-level preferences are composed according to the *Pareto composition* operator $\succ_P$. A disclosure set D_i dominates, according to the Pareto composition, a disclosure set D_j if, for each credential c_l in the portfolio, either $D_i[l] \succ_l D_j[l]$ or $D_i[l] =_l D_j[l]$, meaning that D_i releases a proper subset of the credentials in D_j. For instance, consider the disclosure sets in Table 6, $D_3 \succ_P D_6$ since $D_3[DoB] \succ_{DoB} D_6[DoB]$, that is, $D_6 = D_3 \cup \{\text{DoB}\}$.

The most interesting kind of preferences modeled by the solution in [22] is represented by *amalgamated preferences*, which compare the release of sets of credentials that are not related by a subset-containment relationship. Amalgamated preferences are of the form $c_i \longrightarrow c_j$, meaning that the client prefers to release credential c_i over credential c_j. This preference defines a dominance relationship, denoted $\succ_{\{i,j\}}^{(1,0)(0,1)}$, among disclosure sets. More formally, disclosure set D_k dominates, according to amalgamated preference $c_i \longrightarrow c_j$, disclosure set D_l if $D_k[i] = 1$, $D_k[j] = 0$, $D_l[i] = 0$, $D_l[j] = 1$, and $D_k[x] = D_l[x]$, for all $x \neq i$, $x \neq j$. For instance, consider the disclosure sets in Table 6 and amalgamated preference *Insurance* $\longrightarrow$ *Dialysis*, then $D_{10} \succ_{\{Insurance,Dialysis\}}^{(1,0)(0,1)} D_7$ since they both disclose credentials *VISANum* and *VISALimit*, but D_{10} releases *Insurance* while D_7 releases *Dialysis*. Analogously, $D_{11} \succ_{\{Insurance,Dialysis\}}^{(1,0)(0,1)} D_8$ and $D_{12} \succ_{\{Insurance,Dialysis\}}^{(1,0)(0,1)} D_9$. Note that the binary sub-vectors on the top of the dominance operator can be any pair of binary sub-vectors of the same length. Amalgamated preferences can be conveniently represented through a graph, whose vertexes model credentials and whose edges represent disclosure preferences among them. Note that, to avoid inconsistencies in the definition of privacy preferences, the disclosure graph must be acyclic. The model in [22] permits also to specify conditions associated with preferences, meaning that a dominance relationship holds only if the associated condition is satisfied (e.g., only if a given credential has already been disclosed). For instance, the client may prefer to release credential *Insurance* over her *Name* if credential *InsCoverage* has already been released (since the server is aware of the fact that the client has subscribed an insurance). These conditions are graphically represented by labels associated with the edges of the preference graph. Figure 4 illustrates an example of graph representing amalgamated preferences for the portfolio in our example. Consider the disclosure sets

in Table 6, according to the preferences in the graph, $D_1 \succ^{(1,0)(0,1)}_{\{DoB,City\}} D_4$ and $D_2 \succ^{(1,0)(0,1)}_{\{DoB,City\}} D_5$ since the disclosure of *DoB* is preferred to the disclosure of *City*. Also, $D_{12} \succ^{(1,0)(0,1)}_{\{Insurance,Name\}} D_3$ since credential *InsCoverage* has already been released and therefore the client prefers to release *Insurance* instead of *Name*.

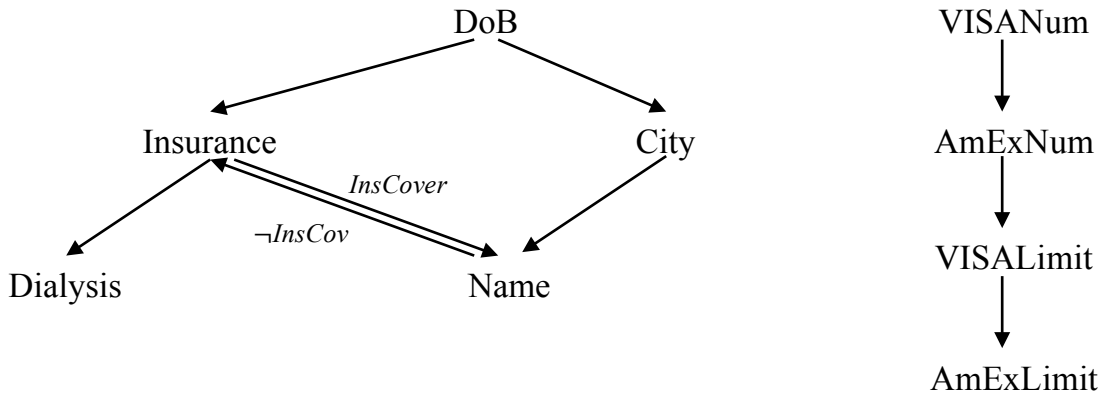

Figure 4 An example of a set of amalgamated preferences

Both the dominance relationship defined by the Pareto composition and the dominance relationships induced by amalgamated preferences permit to compare disclosure sets that differ only for the release of the subset of credentials on which the dominance relationship has been defined. However, it may happen that two disclosure sets cannot be compared considering one dominance relationship only, but they can be compared combining two or more disclosure preferences. For instance, consider the graph in Figure 4 and the disclosure sets in Table 6. Disclosure sets D_1 and D_2 cannot be directly compared, but it is immediate to see that D_1 dominates D_2 by combining amalgamated preferences *VISANum⟶AmExNum* and *VISALimit⟶AmExLimit*. In fact, D_1 discloses the attributes of VISA credit card, while D_2 discloses the attributes of AmEx credit card. In [22] the authors propose to incrementally compose certificate-level and amalgamated preferences. The transitive closure of all the preferences in the system permits to define a *complete preference relationship*, denoted $\succ\succ$, which summarizes all the preference relationships expressed by the client. As a consequence, given the access control policy p regulating the release of the service requested by the client, the approach in [22] permits to limit the set of successful disclosure strategies among which the client needs to choose. In fact, the choice can be restricted to the *optimal disclosure sets*, that is, to the sets of credentials in the client portfolio that satisfy p and that are not dominated by another disclosure set that satisfies p. More formally, the set of optimal disclosure sets is defined as follows [22].

Definition 2: **Optimal Disclosure Sets -** Let $C = \{c_1, \ldots, c_n\}$ be the set of credentials in the client portfolio; s be the service requested by the client; p be the access control policy regulating the release of s; $\mathcal{D} = \{D_1, \ldots, D_m\}$ be the set of disclosure sets that satisfy p, with $D_i \subseteq C$, $i = 1, \ldots, n$; and $\succ\succ$ be a complete preference relationship over C. An optimal disclosure set $\mathcal{D}_{\succ\succ}$ of $\mathcal{D}$ wrt $\succ\succ$ is defined as: $\mathcal{D}_{\succ\succ} = \{D \in \mathcal{D} | \nexists D' \in \mathcal{D}, D' \succ\succ D\}$.

The disclosure sets in $\mathcal{D}_{\succ\succ}$ are optimal and cannot be compared with respect to the disclosure preferences defined by the client (i.e., they are equivalent according to client preferences). To finally decide which set of credentials to disclose to the server, the client needs to choose, among the negotiation strategies in $\mathcal{D}_{\succ\succ}$, the one she prefers to disclose. For instance, with reference to the disclosure sets in Table 6 and the preferences in Figure 4, $\mathcal{D}_{\succ\succ} = \{D_1, D_{10}, D_{12}\}$.

Open Issues. The solution proposed in [22] has the great advantage over the approaches discussed in Section 2 and in Section 3 of modeling and managing qualitative preferences. In fact, it permits to specify privacy preferences at the attribute granularity, and it defines a partial order relationship and different composition operators over the domain of privacy preferences, therefore resulting easy to use for the client. However, it still needs to be enhanced to comply with all the desiderata that a privacy-aware access control system should satisfy (see Section 2). The main shortcoming from which the proposal in [22] suffers is that it requires the client intervention in the choice of the set of credentials to disclose among the successful strategies in the optimal set. Also, the proposed model assumes that each credentials in the client portfolio certifies one attribute only, while often credentials include a set of attributes that cannot be singularly released (i.e., atomic credentials).

6. PRIVACY PREFERENCES IN CREDENTIAL-BASED INTERACTIONS

The first solution that formally models the client portfolio to permit the client to specify fine-grained privacy preferences, as well as constraints on the disclosure of portfolio components, has been proposed in [3]. One of the main advantages of the portfolio modeling in [3] is that it permits to represent both atomic and non-atomic credentials, declarations, and the attributes composing them, clearly distinguishing between credential-dependent and credential-independent attributes. As a consequence, this modeling permits to easily associate privacy preferences with each credential and attribute in the client portfolio. More precisely, the client portfolio is modeled as a bipartite graph $G(V_C \cup V_A, E_{CA})$ with a vertex for each credential and each attribute in the portfolio and an edge connecting each credential to the attributes it

certifies. It is important to note that each credential-independent attribute is represented by a vertex in G, while each credential-dependent attribute is represented by several vertexes (one for each credential certifying it). For instance, Figure 5 illustrates the graph representing the portfolio in Table 1, where we distinguish atomic credentials by attaching all the edges incident to the vertex representing the credential to a black semi-circle. The label of vertexes representing credentials is of the form *id:*type, where *id* is the identifier of the credential and *type* is its type. The label of vertexes representing attributes is of the form `name`:value.

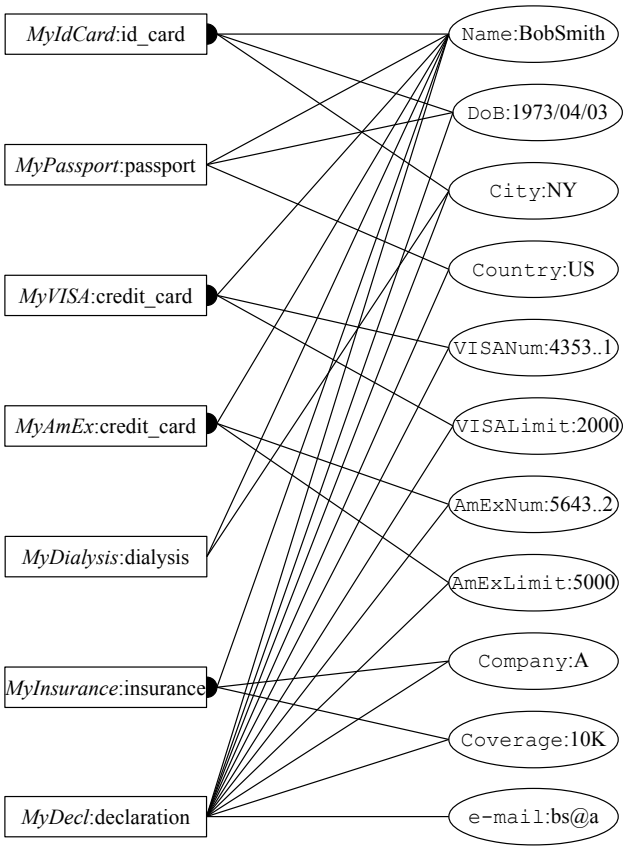

Figure 5 Portfolio graph of the portfolio in Table 1

Sensitivity Labels. The client can define her privacy preferences at a fine-granularity level by associating a *sensitivity label* with each credential and attribute (or combinations thereof) in her portfolio. These labels represent how much the client values the disclosure of the portfolio components. The domain Λ of sensitivity labels can be any set of values characterized by a partial order relationship $\succcurlyeq$, and a composition operator $\oplus$. This generic definition of sensitivity labels captures different

methods for expressing preferences. For instance, sensitivity labels could be positive integer values, where the order relationship $\succcurlyeq$ is the traditional $\geq$ relationship and the composition operator can either be the sum (i.e., +) or the maximum. In the example, for simplicity, we will consider numerical sensitivity labels. Labeling function λ associates a sensitivity label in Λ with each credential c, with each attribute a in the client portfolio, and possibly with subsets thereof. Figure 6 illustrates the portfolio graph in Figure 5, extended by associating each vertex with its sensitivity label and by including new vertexes that represent associations and disclosure constraints. The semantics of the sensitivity labels associated with portfolio components can be summarized as follows.

- $\lambda(a)$: defines the sensitivity of attribute a singularly taken and reflects how much the client values its disclosure. For instance, with reference to the portfolio graph in Figure 6, $\lambda(VISANum) \geq \lambda(DoB)$ since the client considers the number of her VISA more sensitive than her date of birth.
- $\lambda(c)$: defines the sensitivity of the *existence* of credential c. This label reflects how much the client values the additional information carried by the credential itself, independently from the attributes it certifies. For instance, with reference to the portfolio graph in Figure 6, $\lambda(MyDialysis)$ reflects the sensitivity associated by the client with the credential certifying her nephrological disease, independently from the fact that this credential also certifies attributes *Name* and *City*. Clearly, the existence of the credential itself has a sensitivity that goes beyond the demographical information it certifies.

The sensitivity label associated with the combined release of a set of credentials and attributes generally corresponds to the composition through operator $\oplus$ of the sensitivity labels of each portfolio component in the released set. For instance, the release of atomic credential *MyIdCard* has sensitivity label $\lambda(MyIdCard) \oplus \lambda(Name) \oplus \lambda(DoB) \oplus \lambda(City)$. There are however cases where the combined release of some portfolio components may cause a higher or lower information disclosure than the sensitivity label obtained composing the labels of the released credentials and attributes. To capture these situations, the model in [3] permits the client to specify sensitivity labels for subsets of portfolio components, representing how much the client values the release of the *association* of their values. The sensitivity labels of associations must then be considered when composing the sensitivity labels of the attributes and/or credentials in the association. Graphically, associations are represented by additional vertexes in the portfolio graph, connected to the attributes

and/or credentials composing the associations. In particular, the following two kinds of associations are modeled.

- *Sensitive views* model situations where the combined release of a set of portfolio components carries *more* information than the composition of the sensitive labels of its components. For instance, with reference to the portfolio graph in Figure 6, $\lambda(\{DoB,City\})$=4 models the additional sensitivity carried by the combined release of the two attributes.
- *Dependencies* model situations where the combined release of a set of portfolio components carries *less* information than the composition of the sensitive labels of its components. For instance, with reference to the portfolio graph in Figure 6, $\lambda(\{City,Country\})$=-2 represents the sensitivity to be removed when the two attributes are released together, since the knowledge of the *City* where a user leaves permits to easily infer her *Country*. The sensitivity label associated with a dependency $\mathcal{A} = \{c_i, \ldots, c_j, a_k, \ldots, a_l\}$ can assume any value, provided the sensitivity label of the combined release of all the credentials and attributes in $\mathcal{A}$ dominates the sensitivity label of the most sensitive element in $\mathcal{A}$ (i.e.,

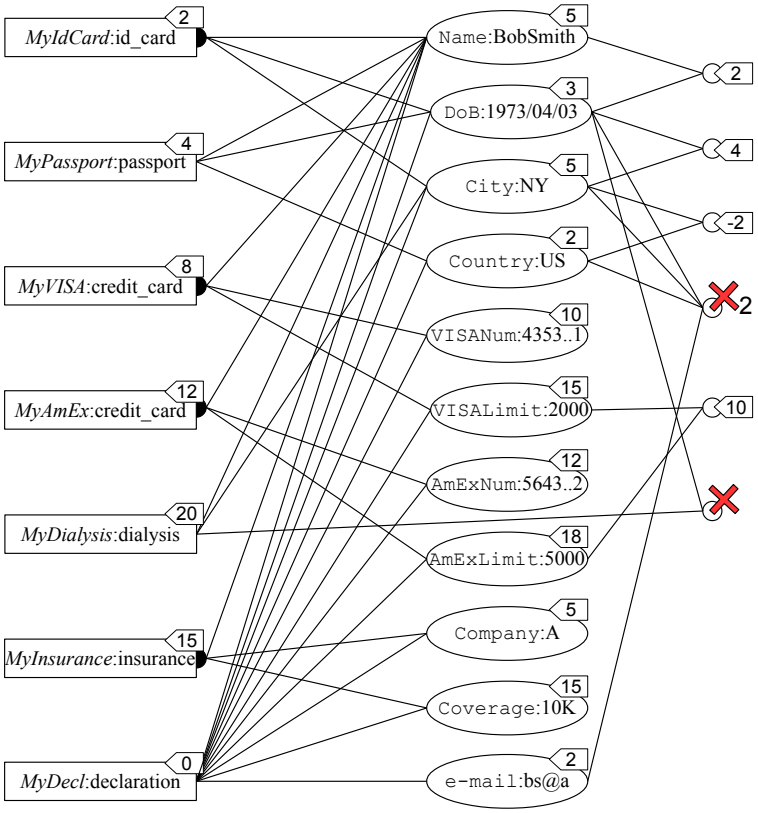

Figure 6 Portfolio graph in Figure 5 extended with sensitivity labels, associations, and constraints

$$\lambda(c_i) \oplus \ldots \oplus \lambda(c_j) \oplus \lambda(a_k) \oplus \ldots \oplus \lambda(a_l) \oplus \lambda(\mathcal{A}) \geqslant \max(\lambda(x),\ x \in \mathcal{A})).$$

In addition to sensitivity labels associated with credentials, attributes, and subsets thereof, the client may need to specify disclosure constraints that cannot be expressed through sensitivity labels. To this purpose, in [4] the authors extend the original model introduced in [3] with the following two kinds of constraints.

- *Forbidden views* represent subsets of portfolio components whose combined release is prohibited. For instance, with reference to the portfolio graph in Figure 6, forbidden view {*DoB*,*MyDialysis*} prevents the combined release of attribute *DoB* and credential *MyDialysis* and is graphically represented by a cross-shaped vertex connected with the attribute and credential in the constraint.
- *Disclosure limitations* represent subsets of portfolio components characterized by restrictions of the form *at most n* elements in the set can be jointly disclosed. For instance, with reference to the portfolio graph in Figure 6, disclosure limitation $\{Name,City,Country,e\text{-}mail\}_2$ permits to release at most two attributes in the set and is graphically represented by a cross-shaped vertex with label 2 and connected with all the attributes in the set.

Disclosure. Given the client portfolio, it is important to note that not all the subsets of portfolio components represent a valid disclosure, that is, not all the sets of credentials and attributes can be communicated to the server to gain access to the requested service. First of all, a subset D of portfolio components represents a *disclosure* only if it satisfies the following three conditions.

1. *Certifiability*: each disclosed attribute is certified by at least a credential, whose existence is disclosed as well (i.e., $\forall a \in D, \exists c \in D\ s.t.\ a \in attributes(c)$).
2. *Atomicity*: if an attribute certified by an atomic credential is disclosed, all the attributes in the credential are disclosed (i.e., $\exists c \in D\ s.t.\ c$ is atomic, $\forall a \in attributes(c), a \in D$).
3. *Association exposure*: if all the attributes and/or credentials composing an association are disclosed, then the association itself is disclosed (i.e., $\forall x \in \mathcal{A}, x \in D$ then $\mathcal{A} \in D$).

These conditions permit to easily take into account both atomic and non-atomic credentials, as well as associations, in the computation of the sensitivity label characterizing the disclosure of a set of credentials and attributes. For instance, consider the portfolio graph in Figure 6. An example of disclosure D is represented in

Figure 7(a), where released elements are reported in black while non-released elements are reported in gray. Figure 7(b) represents instead a subset of the portfolio components that does not represent a disclosure, since it violates the above properties. The *sensitivity* of a disclosure D is computed by composing the sensitivity label of all the credentials, attributes, and associations composing it. For instance, the sensitivity of the disclosure in Figure 7(a) is $\lambda(D) = \lambda(MyPassport) + \lambda(MyVISA) + \lambda(MyDecl) + \lambda(Name) + \lambda(DoB) + \lambda(VISANum) + \lambda(VISALimit) + \lambda(e\text{-}mail) + \lambda(\{Name,DoB\}) = 4 + 8 + 0 + 5 + 3 + 10 + 15 + 2 + 2 = 49$. A disclosure is said to be *valid* if it does not violate disclosure constraints. Only valid disclosures can be released. For instance, the disclosure in Figure 7(a) is valid, while the one in Figure 7(c) is not valid since it violates forbidden view {*DoB*,*MyDialysis*}.

Given the server policy p regulating the disclosure of the service of interest, it is necessary to determine a *minimum disclosure* (i.e., a valid disclosure with minimum sensitivity label) satisfying p. In [3], the authors assume that server policies are formulated as Boolean formulas composed of terms of the form $t.\{a_i, \ldots, a_j\}$ in disjunctive normal form. A clause $t.\{a_i, \ldots, a_j\}$ in the server policy requires the disclosure of a credential c of type t that certifies attributes $\{a_i, \ldots, a_j\}$. A valid disclosure D satisfies a term $t.\{a_i, \ldots, a_j\}$ if $\exists c \in D\ s.t.\ type(c) \leq_{isa} t$ and $\{a_i, \ldots, a_j\} \subseteq attributes(c)$. For instance, assume that the policy regulating access to the *MedicineBooking* service is *id*.{*Name*} $\wedge$ *credit_card*.{*Name*,*Number*,*Limit*} $\wedge$ *.{*DoB*,*e-mail*}. The disclosure in Figure 7(a) satisfies the policy and grants the client access to the requested service: term *id*.{*Name*} is satisfied by the release of attribute *Name* from credential *MyIdCard*; term *credit_card*.{*Name*,*Number*,*Limit*} is satisfied by the release of atomic credential *MyVISA*; and term *.{*DoB*,*e-mail*} is satisfied by the release of attribute *DoB* from credential *MyIdCard* and by the declaration of attribute *e-mail*. Formally, the minimum disclosure problem can then be formulated as follows.

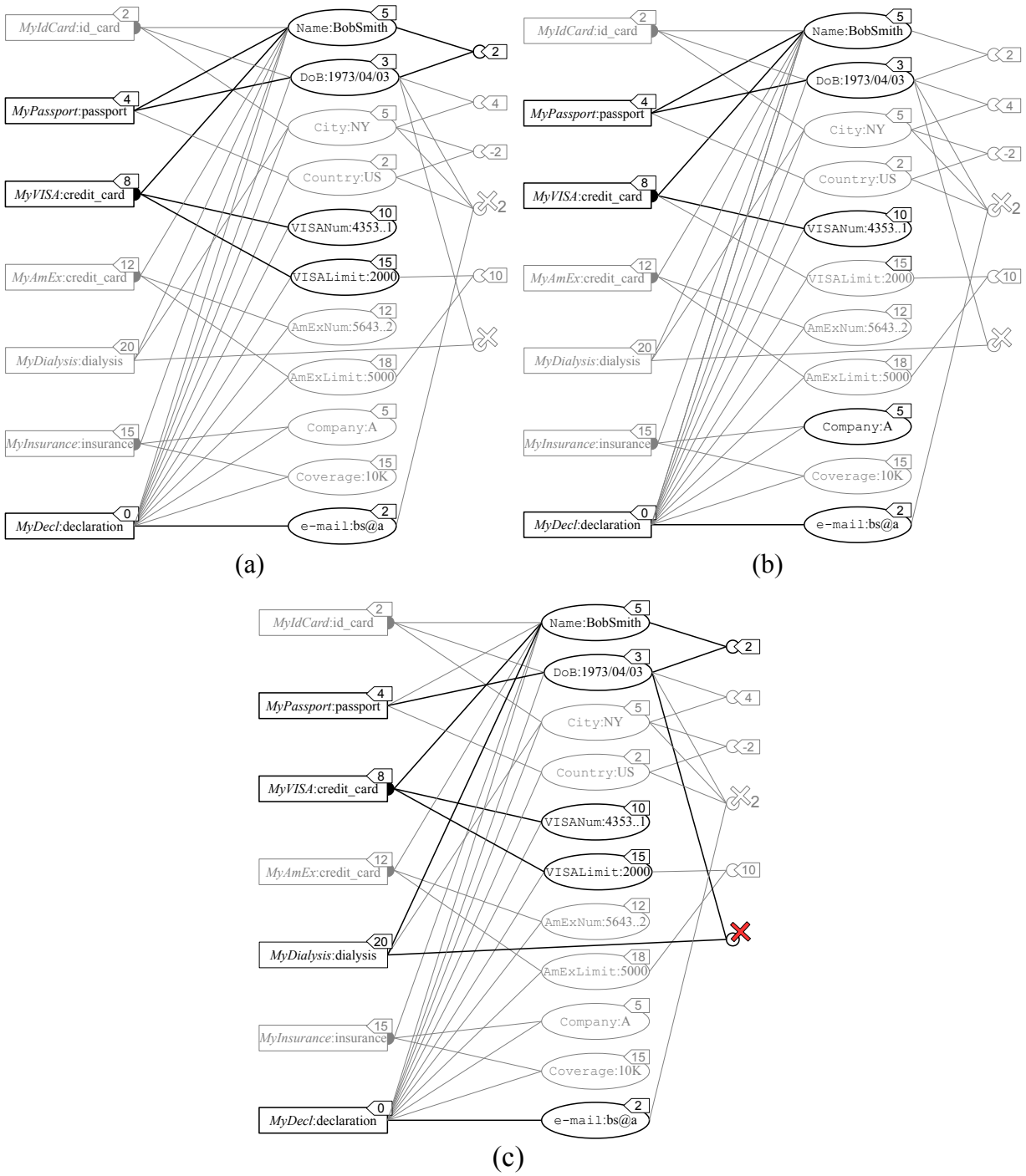

Figure 7 An example of valid disclosure (a), arbitrary subset of portfolio elements (b), and non valid disclosure (c)

Problem 3: **Minimum Disclosure problem -** Let $C = \{c_1, \ldots, c_n\}$ be the set of credentials in the client portfolio; $A = \{a_1, \ldots, a_m\}$ be the set of attributes in the client portfolio; $(T, \leq_{isa})$ be the hierarchy of credential types; $\mathbb{A}$ be the set of sensitive associations; $\mathbb{F}$ be the set of forbidden views; $\mathbb{L}$ be the set of disclosure limitations; λ be the labeling function; and p be the server policy. Find a subset $D \subseteq C \cup A$ such that:

1. $\forall a \in D, \exists c \in D \; s.t. \; a \in attributes(c)$ (certifiability);
2. $\exists c \in D \; s.t. \; c$ is atomic, $\forall a \in attributes(c), a \in D$ (atomicity);
3. $\forall x \in \mathcal{A}, x \in D$ then $\mathcal{A} \in D$(association exposure);
4. $\forall f \in \mathbb{F}, f \nsubseteq D$ (forbidden views satisfaction);
5. $\forall l_i \in \mathbb{L}, \nexists l' \subseteq D \; s.t. \; |l'| \geq i$, with i the threshold fixed by constraint l_i (disclosure limitation satisfaction);
6. $\exists$ a clause $t_1.\{a_{i1}, \ldots, a_{j1}\} \wedge \ldots \wedge t_l.\{a_{il}, \ldots, a_{jl}\}$ in p such that for each term $t.\{a_i, \ldots, a_j\}$ in the clause, $\exists c \in D \; s.t. type(c) \leq_{isa} t$ and $\{a_i, \ldots, a_j\} \subseteq attributes(c)$ (policy satisfaction);
7. $\nexists D'$ satisfying all the conditions above and such that $\lambda(D) \succ \lambda(D')$.

For instance, the disclosure in Figure 7(a) represents a minimal disclosure for our example.

The problem of computing a minimal disclosure is NP-hard [3]. In [3] the authors propose a graph-based heuristic algorithm to compute a minimal disclosure (i.e., a disclosure that, although not minimal, has a low sensitivity label). In [4] the authors define a modeling of the problem as an instance of the Max-SAT problem, and use Max-SAT solvers to compute an optimum solution in a limited computational time.

The model in [3] has been extended in [6] to permit the client to complement her privacy preferences with *context-based restrictions* that limit the disclosure of credentials on the basis of the context of her request. In the same paper, the authors also propose to take the *history* of past interactions into account in the choice of the set of credentials and attributes to disclose for gaining access to the requested service.

Open Issues. The modeling of the client portfolio proposed in [3] permits the client to specify sensitivity labels at the attribute granularity level and to take advantage of new constructs for taking sensitive associations and disclosure constraints into consideration in the choice of the set of portfolio components to disclose. This approach leaves however space to further improvements. Sensitivity labels modeling privacy preferences may not be easy to define for final users. In fact, as already noted in [22], it is hard to associate a quantitative value with each portfolio component (and possible subset thereof), while it would be easier to define a partial order relationship between subsets of portfolio components.

7. FINE-GRAINED DISCLOSURE OF SENSITIVE ACCESS POLICIES

In [2], the authors address the problem of regulating the disclosure of access control policies, by proposing a model that permits the server to specify a *disclosure policy*

regulating if and how an access control policy should be communicated to the client. To this purpose, the approach in [2] models access control policies as *policy trees*. Policy tree $T(\mathrm{N})$ representing policy p has a node for each operator, attribute, and constant value in p. The internal nodes of the tree represent operators, whose operands are represented by the sub-trees rooted at its children. For instance, Figure 8 represents the policy tree of p = $(type(c_1)$ = *credit_card* ∧ c_1.*Limit* > 1,000) ∨ $(type(c_2)$ = *insurance* ∧ c_2.*Company* ≠ 'A' ∧ c_2.*Company* ≠ 'B').

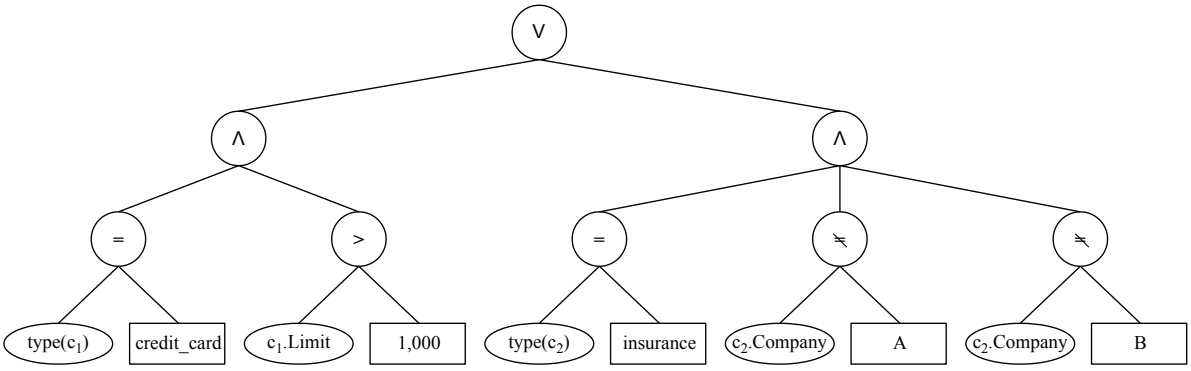

Figure 8 An example of policy tree

Disclosure Policy. The disclosure policy regulating the release of a policy p to a client regulates the visibility of each node in the policy tree $T(\mathrm{N})$. The disclosure policy is formally defined as a coloring function $\gamma: N \rightarrow \{green, yellow, red\}$ that associates with each node n in the policy tree a color in the set $\{green, yellow, red\}$, thus obtaining a *colored policy tree* $T(\mathrm{N}, \gamma)$. The semantics of the colors, with respect to the client visibility of a node, can be summarized as follows:

- $green$: the node is released;
- $yellow$: the label of the node is removed (i.e., the operator, attribute, or constant value it represents) before its release, while its presence in the tree and its children are preserved;
- red: the label of the node is removed and possibly also its presence in the tree.

As an example, Figure 9 illustrates a possible coloring regulating the disclosure of the policy tree in Figure 8. In the figure, $green$ nodes are white, $yellow$ nodes are gray, and red nodes are black.

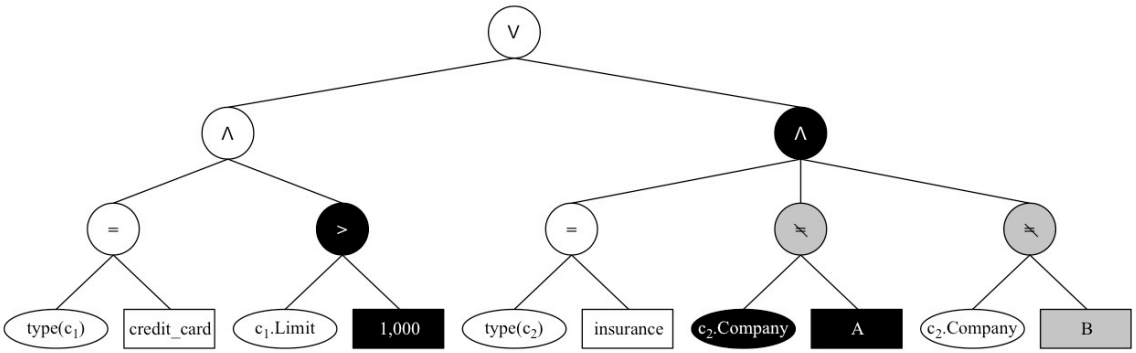

Figure 9 An example of coloring for the policy tree in Figure 8

Although the server can decide to associate an arbitrary color with each node in the tree, a disclosure policy is *well defined* if it is meaningful. More precisely, a disclosure policy is well defined if it satisfies the following conditions.

1. If a leaf node representing a constant value is $green$, then its sibling (which represents an attribute) is $green$ and its parent (which represents an operator) is not red.
2. If a node representing an operator is $green$, at least one of its children must be either $green$ or $yellow$.
3. The nodes in a sub-tree representing a condition on credential type must be either all $green$ or all red.

Figure 10 illustrates an example of non well defined coloring for the policy tree in Figure 8. In fact, since only node 'A' is $green$ in the sub-tree representing condition $c_2.Company \neq$ 'A', the server would disclose value 'A' instead of the condition. Analogously, only node > is $green$ in the sub-tree of condition $c_1.Limit >$ 1,000, the server would disclose the operator > only to the client. Finally, node *insurance* is $yellow$ in the sub-tree of condition $type(c_2) =$ *insurance*, while the other nodes are $green$. The disclosed condition would then only release operand $type(c_2)$ and operator =, which does not give to the client any information to possibly gain access to the service.

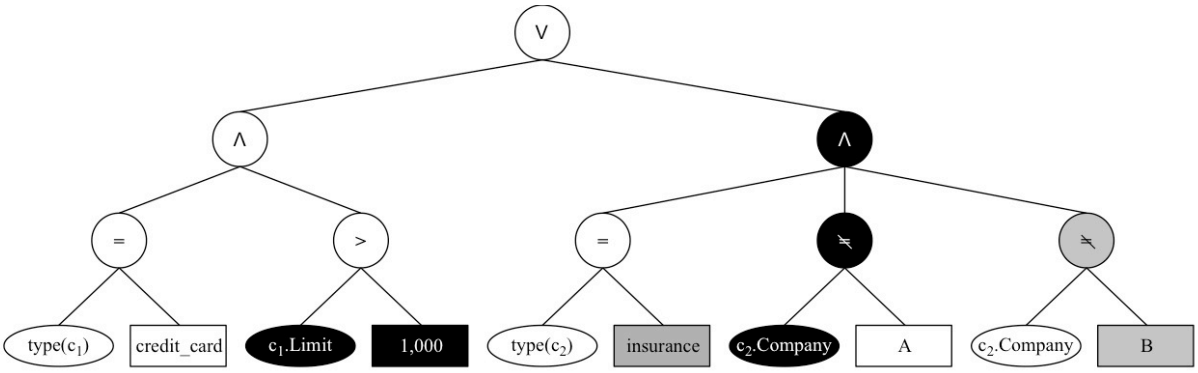

Figure 10 An example of non well defined colored policy tree

Policy Communication. When the client sends a request for accessing a service to the server, the server transforms its access control policy into a client policy view according to the disclosure policy. The colored policy tree $T(\mathrm{N}, \gamma)$ regulating policy disclosure is therefore transformed into an equivalent *client policy tree view* by: *i)* removing the label of $yellow$ and red nodes; *ii)* removing unnecessary red leaves; and *iii)* collapsing internal red nodes in a parent-child relationship in a single red node. To this purpose, the server visits the tree following a post-order strategy and applies, in the order, the following three classes of transformation rules.

- *Prune rules*. These rules remove unnecessary leaf nodes. Two kinds of prune rules can be applied on an internal node n whose children are leaf nodes.
  - *Red predicate rule*. If n is red, all its red children are removed. For instance, consider the colored policy tree in Figure 9, according to this rule node representing constant value 1,000 is removed.
  - *Red children rule*. If all the children of n are red, they are removed and the color of node n is set to red. For instance, consider the colored policy tree in Figure 9, according to this rule the nodes representing attribute *Company* and constant value 'A' in condition ($c_2.Company \neq$ 'A') are removed. Also, the color of the node representing operator $\neq$ is set to red.
- *Collapse rule*. This rule operates on internal red nodes and removes their non-leaf red children. For instance, consider the colored policy tree in Figure 9, the node representing operator $\neq$ in condition ($c_2.Company \neq$ 'A') is removed since its parent (i.e., the second child of the root node) is red.
- *Hide label rule*. This rule removes the labels of $yellow$ and red nodes.

Figure 11 illustrates the client policy tree view obtained applying the transformation rules described above to the colored policy tree in Figure 9.

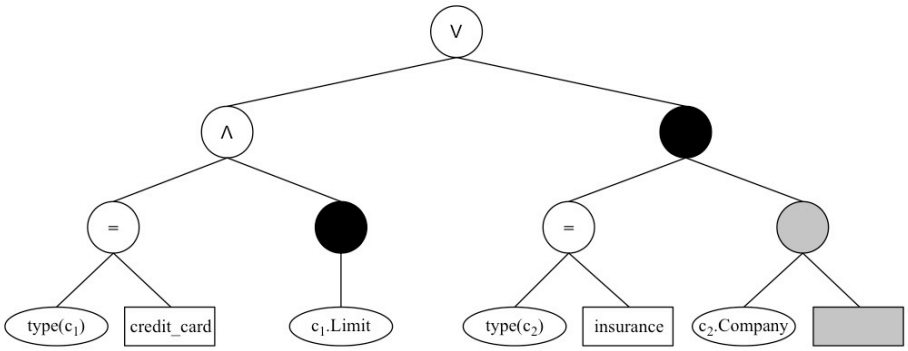

Figure 11 Policy tree view of the colored policy tree in Figure 9

The disclosure of a client policy tree view may be meaningless for the client, since it may not represent in a "fair way" the server access control policy [2]. Intuitively, a client policy tree view fairly represents the server policy if it includes at least a subset of attributes that permit the access control policy evaluation. In fact, in this case, the client can decide whether to release the requested attributes to possibly gain access to the service of interest. Clearly, the server should disclose only fair policies. For instance, the policy view represented by the tree in Figure 11 is fair, since all the attributes and credential types in the original policy are preserved in the client view. The client can decide whether to release either one of her credit cards or her insurance to possibly gain access to the *MedicineBooking* service.

Open Issues. The solution proposed in [2] to protect the confidentiality of access control policies, while permitting the client-server interaction in open environments, is effective and permits the definition of disclosure restrictions at a fine granularity level. However, this proposal represents only a first step in the definition of an effective system regulating policy disclosure. In fact, the proposed model permits to check whether a disclosure policy generates a fair client policy tree view but it does not propose an approach for possibly revising the disclosure policy when the client policy tree view is not fair (and therefore prevents the definition of a successful negotiation strategy). Also, the model could be extended to consider the disclosure of proofs of possession and/or proofs of satisfaction of condition.

8. OPEN ISSUES

The enforcement of access privileges in open environments taking into account both client privacy preferences and policy confidentiality requirements still present different open issues that need to be addressed.

- *Inheritance of privacy preferences*. Most of the solutions proposed in the literature assume that the client associates a preference with each credential

and/or attribute in her portfolio. Although the solution in [3] uses the hierarchy of credential types for checking whether the disclosure of a subset of the portfolio components satisfies a given server request, it does not consider this hierarchy in the definition of privacy preferences. An interesting open issue consists in exploiting the hierarchy of credential types to make the definition of privacy preferences more user-friendly.

- *Proof of possession*. The values modeling privacy preferences are traditionally associated with credentials and/or attributes and express how much their owner values their release. Recent technologies however permit to release proofs of possession of credentials and proofs of satisfaction of conditions defined on attributes. The release of a proof is usually considered less sensitive than the release of the credential/attribute on which the proof is based. This different disclosure risk should therefore be adequately modeled.
- *Shared knowledge*. Attribute-based access control solutions traditionally assume that the hierarchy of credential types and attribute names represent a common knowledge for the server and the client. However, this assumption does not always hold in real-life scenarios, where there may be mismatches due also to the fact that servers refer to credential and attribute types while clients refer to their instances. Access control models should be extended to handle this problem.
- *Integration*. Both the solutions developed to support client privacy preferences and the solutions proposed to protect the confidentiality of server policies do not consider the privacy requirements of the counterpart. It is therefore important to study new models that consider both the client and the server privacy needs.
- The approaches proposed in the literature for the support of client privacy preferences present advantages and disadvantaged complementary to each other. For instance, the solution in [22] has the advantage of usability, while the approach in [3] supports sensitive associations and disclosure constraints. An interesting open issue is therefore the definition of a model that combines the advantages of all the proposed approaches.

9. SUMMARY

We have analyzed the privacy issues that may arise in open scenarios, where the client accessing a service and the server offering it may be unknown to each other and need to exchange information to build a trust relationship. We have illustrated both the problem of taking client privacy preferences into account in credential disclosure, and the problem of maintaining the confidentiality of server access control policies. For

each of these problems, we have described some recent approaches for their solution and illustrated some open issues that still need to be addressed.

ACKNOWLEDGMENTS

We would like to thank Sabrina De Capitani di Vimercati for suggestions and comments on the chapter organization and presentation. This work was partially supported by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4).

REFERENCES

[1] C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Trust Management," in M. Petkovic, W. Jonker (Eds.), *Security, Privacy and Trust in Modern Data Management*, Springer-Verlag, 2007.

[2] C. A. Ardagna, S. De Capitani di Vimercati, S. Foresti, G. Neven, S. Paraboschi, F.-S. Preiss, P. Samarati, M. Verdicchio, "Fine-grained disclosure of access policies," in *Proc. of ICICS 2010*, Barcelona, Spain, December 2010.

[3] C. A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, P. Samarati, "Minimizing disclosure of private information in credential-based interactions: A Graph-based approach," in *Proc. of PASSAT 2010*, Minneapolis, MN, USA, August 2010.

[4] C. A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, P. Samarati, "Supporting privacy preferences in credential-based interactions," in *Proc. of WPES 2010*, Chicago, IL, USA, October 2010.

[5] C. A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, P. Samarati, "Supporting user privacy preferences on information release in open scenarios," in *Proc. of the W3C Workshop on Privacy and Data Usage Control*, Cambridge, MA, USA, October 2010.

[6] C. A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, P. Samarati, "Minimising disclosure of client information in credential-based interactions," in *IJIPSI*, 1(2/3): 205-233, 2012.

[7] C. A. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, M. Verdicchio, "Expressive and deployable access control in open Web service applications," in *IEEE TSC*, 4(2):6-109, April-June 2011.

[8] A. Armando, A. Contento, D. Costa, M. Maratea, "Minimum disclosure as Boolean optimization: New results," in *Proc. of the 19th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, Rome, Italy, June 2012.

[9] P. Bonatti, P. Samarati, “A uniform framework for regulating service access and information release on the Web,” in *JCS*, 10(3):241-272, 2002.

[10] S. Brands, “Rethinking public key infrastructure and digital certificates - building in privacy,” *MIT Press*, 2000.

[11] J. Camenisch, A. Lysyanskaya, “An efficient system for non-transferable anonymous credentials with optional anonymity revocation,” in *Proc. of EUROCRYPT 2001*, Innsbruck, Austria, May 2001.

[12] W. Chen, L. Clarke, J. Kurose, D. Towsley, “Optimizing cost-sensitive trust-negotiation protocols,” in *Proc. of INFOCOM 2005*, Miami, FL, USA, March 2005.

[13] S. Cimato, M. Gamassi, V. Piuri, R. Sassi, F. Scotti, “Privacy-aware biometrics: Design and implementation of a multimodal verification system,” in *Proc. of ACSAC 2008*, Anaheim, CA, USA, December 2008.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “Introduction to algorithms,” Second edition, *MIT Press*, 2001.

[15] I. Damgrad, M. Jurik, “A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system,” in *Proc. of PKC 2001*, Cheju Island, Korea, February 2001.

[16] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Psaila, P. Samarati, “Integrating trust management and access control in data-intensive web applications,” in *ACM TWEB*, 6(2): 6:1-6:43 , May 2012.

[17] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, P. Samarati, “Access control policies and languages in open environments," in T. Yu, S. Jajodia (Eds.), *Secure Data Management in Decentralized Systems*, Springer-Verlag, 2007.

[18] S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, P. Samarati, “Trust management services in relational databases," in *Proc. of ASIACCS 2007*, Singapore, March 2007.

[19] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati, “Access control,” in H. Bidgoli (Ed.), *The Handbook of Computer Networks*, Wiley, 2008.

[20] M. Gamassi, V. Piuri, D. Sana, F. Scotti, “Robust fingerprint detection for access control,” in *Proc. of RoboCare 2005*, Rome, Italy, May 2005.

[21] K. Irwin, T. Yu, “Preventing attribute information leakage in automated trust negotiation,” in *Proc. of ACM CCS 2005*, Alexandria, VA, USA, November 2005.

[22] P. Kärger, D. Olmedilla, W.-T. Balke, “Exploiting preferences for minimal credential disclosure in policy-driven trust negotiations,” in *Proc. of SDM 2008*, Atlanta, GA, USA, August 2008.

[23] A. J. Lee, M. Winslett, J. Basney, V. Welch, “The Traust authorization service,” in *ACM TISSEC*, 11(1):1-33, February 2008.

[24] J. Li, N. Li, and W.H. Winsborough, “Automated trust negotiation using cryptographic credentials,” in *Proc. of ACM CCS 2005*, Alexandria, VA, USA, November 2005.

[25] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proc. of EUROCRYPT 1999*, Prague, Czech Republic, May 1999.
[26] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, K.E. Seamons, “Adaptive trust negotiation and access control,” in *Proc. of SACMAT 2005*, Stockholm, Sweden, June 2005.
[27] P. Samarati, “Protecting respondents’ identities in microdata release," in *IEEE TKDE*, 13(6): 1010-1027, November/December, 2001
[28] P. Samarati, S. De Capitani di Vimercati, “Access control: Policies, models, and mechanisms,” in R. Focardi, R. Gorrieri (Eds.), *Foundations of Security Analysis and Design*, Vol. 2171 of LNCS, Springer-Verlag, 2001.
[29] R. Sandhu, P. Samarati, “Authentication, access control and intrusion detection,” in A. Tucker (Ed.), *CRC Handbook of Computer Science and Engineering*, CRC Press Inc., 1997.
[30] K. E. Seamons, M. Winslett, T. Yu, “Limiting the disclosure of access control policies during automated trust negotiation,” in *Proc. of NDSS 2001*, San Diego, CA, USA, April 2001.
[31] W. Winsborough, K. E. Seamons, V. Jones, “Automated trust negotiation,” in *Proc. of DISCEX 2000*, Hilton Head Island, SC, USA, January 2000.
[32] D. Yao, K. B. Frikken, M. J. Atallah, R. Tamasia, “Private information: To reveal or not to reveal,” in *ACM TISSEC*, 12(1):1-27, October 2008.
[33] T. Yu, M. Winslett, “A unified scheme for resource protection in automated trust negotiation,” in *Proc. of the IEEE Symposium on Security and Privacy 2003*, Berkeley, CA, USA, May 2003.
[34] T. Yu, M. Winslett, K. E. Seamons, “Supporting structured credentials and sensitive policies trough interoperable strategies for automated trust,” in *ACM TISSEC*, 6(1):1-42, February 2003.