Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

DS2OS – Deliverable C Implementing Directory Services

Giovanni Schmid, Luca Tamburo

RT-ICAR-NA-2010-04 **Novembre 2010**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

DS2OS – Deliverable C Implementing Directory Services

Giovanni Schmid[1], Luca Tamburo

Rapporto Tecnico N.:
RT-ICAR-NA-2010-04

Data:
Novembre 2010

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, Via P. Castellino 111, 80131 Napoli

DS2OS - Deliverable C
Implementing Directory Services

Giovanni Schmid, Luca Tamburo

November 15, 2010

1 Introduction

As detailed in DS2OS-Deliverable A, a DS2OS environment adopts the dynamic delegation (DD) access control model to enconmpass the requirements and objectives of its global security policy. The scope of this Deliverable is to specifically discuss issues and design choices related to DD enforcement at the network layer. These issues and design choices concern two different although related aspects: (i) the use and specification of certificates according to DD requirements, and; (ii) the implementation of an LDAP service which allows the enforcement of the DD model in a DS2OS environment.

A DS2OS environment can be composed of many heterogeneous hosts, having their own local users and access control policies. For efficiency and scalability, the authentication and authorization information which compose a principals profile should be distributed among network-wise and local-only repositories. Only the information which is relevant to multiple hosts should be exported at the network level, whilst the host-specific information should be stored and managed through the hosts local repositories.
Since we have adopted X.509 certificates to share authentication and authorization information at the network level, and since local information is managed through host-specific files and formats, an integration is required between the network and the host layers in order to retrieve the correct security contexts that a given principal running a task must hold on each of the hosts involved in such a task.
The focus of our implementation is on the OpenSolaris (or, equivalently, Solaris 10) OS[1], since at the moment - as throughfully discussed in Deliverable B - these are the general-purpose OS platfoms with the most advanced access control features. In particular, they are the only platforms supporting access control policies which include the management of authorizations directly at the kernel level, virtually for any process in the user space. This special authorizations, called *process privileges*, are very important for an effective enforcement of the DD model, since they allows for the specification of principal's capabilities at an unprecedented fine-grain level, and with much greater flexibility.

This document is organized as follows. Section 2 introduce to X.509 certificates and their related management infrastructures. In Section 3, we discuss Lightweight Directory Access Protocol (LDAP), and its use for managing both basic user-related information and the extended user security attributes introduced with OpenSolaris. Finally, Section 4 illustrates our enforcement of DD using X.509 certificates and OpenLDAP [1], a major open source implementation of the LDAP protocol.

[1]For the sake of simplicity, as in other DS2OS deliverables, we will use the term OpenSolaris to refer in general to Solaris-related technologies

2 X.509 Certificates and Infrastructures

The ITU-T X.509 standard specifies a directory service to implement authentication and authorization based on public key cryptography.
The standard defines two types of certificates: **Public Key Certificate** (**PKC**), and **Attribute Certificate** (**AC**). X.509 PKCs and ACs are the most relevant and adopted way to convey with integrity protection authentication and authorization information over open networks and distributed environments. In order to achieve that, however, ad-hoc infrastructures must be deployed for the distribution and management of certificates. The scope of this section is to introduce to X.509 certificates and their related management infrastructures. Section 2.1 and 2.2 illustrates PKCs and ACs, respectively. In Section 2.3 we do a comparison between the infrastructures for PKC management and those for the management of ACs, showing their similarities and mutual interactions. Finally, Section 2.4 discusses the various models provided by the X.509 standard to manage privileges.

2.1 Public Key Certificates

A PKC is an electronic document which uses a digital signature to bind together a public key with a principal's identity.
In the X.509 system, a **Certification Authority** (**CA**) issues a certificate binding a public key to a particular **distinguished name**; the CA uses its **private key** to digitally sign a PKC.
Let `A` and `Ap` be the **distinguished name** and a **public key** for a given principal, respectively. If `CA` is the name of a CA, then the cleartext part of a PKC issued by `CA` to `A` has the form:

$$\texttt{CA<<A>> = CA\{V,SN,AI,CA,UCA,A,UA,Ap,T\}} \ ,$$

where `V` is the certificate version, `SN` its serial number, `AI` is the identifier of the algorithm used to sign the certificate, and `T` is the certificate validity period, in the format "`notBefore` and `notAfter`" [2]. The fields `UCA` and `UA` are optionals; they represent (unique) identifiers for the CA and the subject principal, respectively.
A PKC is represented in ASN.1 syntax as follows:

```
Certificate ::=  SIGNED { SEQUENCE {
  version                 [0]  Version DEFAULT v1,
  serialNumber            CertificateSerialNumber,
  signature               AlgorithmIdentifier,
  issuer                  Name,
  validity                Validity,
  subject                 Name,
  subjectPublicKeyInfo    SubjectPublicKeyInfo,
  issuerUniqueIdentifier  [1] IMPLICIT UniqueIdentifier OPTIONAL,
     -- if present, version shall be v2 or v3
  subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL,
     -- if present, version shall be v2 or v3
  extensions              [3]  Extensions OPTIONAL
     -- If present, version shall be v3
}}
```

The `signature` in the PKC is used to verify the validity of the PKC using the CA's public key. The `validity` of the certificate is the period of time during which the CA will publish any eventual certificate revocation, warranting the status information of the certificate.

There are mainly two types of PKC, the **end–entity certificate** and the **CA–certificate**. An end-entity certificate is a PKC issued to a subject which does not issue other PKCs, whilst a CA–certificate is a PKC issued by a CA to another CA. In turn, a CA–certificate can be categorized as one of the following types:

- **Self–issued certificate**, in which the issuer and the subject are the same.
- **Self–signed certificate**, which is a special case of self-issued certificate. The private key used by the CA to sign the certificate corresponds to the public key `Ap` in the certificate.
- **Cross certificate**, where the issuer and the subject are two different CAs. A CA can issue a certificate for another CA (the subject CA) in two ways: in a strict hierarchy of CAs, in order to authorize the subject CA's existance, or to recognize the existence of the subject CA in a distributed trust model.

2.2 Attribute Certificates

In the context of X.509 standard, **privileges** can generically represent authorizations in accessing some resource, roles, or other capabilities given by some authority to a principal[2].

ACs are used to assign privileges to the certificate **holder** by the certificate **issuer**, that can eventually act as a delegated authority (see Section 2.4). Privileges are stored in ACs as set of **attributes** of different kind. ACs are thus a means to convey with integrity protection a permission for the subject principal w.r.t. a service or a resource that the issuer controls in some way.

> Some people constantly confuse PKCs and ACs. An analogy may make the distinction clear. A PKC can be considered to be like a passport: it identifies the holder, tends to last for a long time, and should not be trivial to obtain. An AC is more like an entry visa: it is typically issued by a different authority and does not last for as long a time. As acquiring an entry visa typically requires presenting a passport, getting a visa can be a simpler process [3].

AC and PKC data structures of course differ. However, similarly to a PKC, the set of data constituting an AC is signed by the issuer, called **Attribute Authority** (**AA**), in order to ensure data integrity. The following ASN.1 syntax represents an AC:

```
AttributeCertificate ::= SIGNED{ AttributeCertificateInfo }

AttributeCertificateInfo ::= SEQUENCE {
  version                 AttCertVersion, -- version is v2
  holder                  Holder,
  issuer                  AttCertIssuer,
  signature               AlgorithmIdentifier,
  serialNumber            CertificateSerialNumber,
  attrCertValidityPeriod  AttCertValidityPeriod,
  attributes              SEQUENCE OF Attribute,
  issuerUniqueID          UniqueIdentifier OPTIONAL,
  extensions              Extensions OPTIONAL
}
```

[2]Be aware that the term "privilege" is used with a different, more specific meaning in the context of Solaris security, too; see Deliverable B.

The fields `version`, `signature`, `serialNumber`, `attrCertValidityPeriod` and `issuerUniqueID` have the same meaning of their analogous in a PKC. The `holder` and `issuer` fields represent the subject principal and AA of the AC, respectively. These two fields can have one of the following three forms [2]:

- `GeneralNames`, which identifies one or more names for the entity;
- `baseCertificateID`, which identifies a particular PKC and is used to link the AC to a PKC;
- `objectDigestInfo`, that is used to authenticate directly the identity of the holder by comparing a digest of the corresponding information.

Finally, the `attributes` field contains a set of attributes which establishes the privileges given by the issuer to the holder.

2.3 Public Key vs. Privilege Management Infrastructures

The main purpose of a **Public Key Infrastructure** (**PKI**) is to enable the safe and efficient acquisition of public keys over open networks.

> A PKI is a set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke public-key certificates [4].

The IETF has its own workgroup, the **Public Key Infrastructure X.509** (**PKIX**) working group, which is devoted to the specification of X.509-compliant structures and procedures of valuable usage in the Internet. A PKIX infrastructure defines the elements and functions for PKCs management as specified by the PKIX workgroup. A scheme of the PKIX model is illustrated in Figure 1.

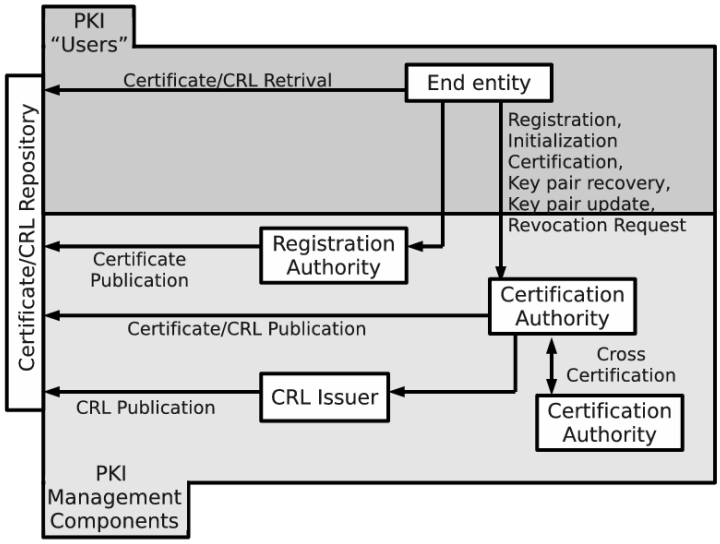

Figure 1: PKIX model.

Concept	PKI Entity	PMI Entity
Certificate	Public Key Certificate	Attribute Certificate
Certificate issuer	Certification Authority	Attribute Authority
Certificate user	Subject	Holder
Certificate binding	Subject's name to public key	Holder's name to privilege attribute(s)
Revocation	Certificate revocation list (CRL)	Attribute certificate revocation list (ACRL)
Root of trust	Root certification authority or trust anchor	Source of authority
Subordinate authority	Subordinate certification authority	Attribute authority

Table 1: A comparison of PKIs with PMIs [5].

A **Privilege Management Infrastructure** (**PMI**) is very similar to a PKI (see table 1), and it is used for the management of Attribute Certificates. Both PKIs and PMIs are represented by tree structures rooted at a primary source of trust, and having intermediate sources of trust as nodes and end-users as leaves. A PKI root of trust is sometimes called **root CA**, while the root of trust of a PMI is called **Source of Authority** (**SOA**). A CA may have subordinate CAs (its children in the tree), that it delegates to sign PKCs; similarly, SOAs may delegate their power to subordinate AAs. If a principal needs to have its public key revoked, a CA will issue a **Certificate Revocation List** (**CRL**); similarly, if a principal needs to have its privileges revoked, an AA will issue an **Attribute Certificate Revocation List** (**ACRL**)[5].

Usually PMIs relies on an underlying PKI, since ACs have to be digitally signed by the issuing AA, and the PKI is used to validate the AA's signature. A joint adoption of PKI and PMI is suitable in environments where any one of the following is true:

- Different authorities are responsible for issuing authentic public keys and assigning privileges to the same subjects;
- There are a number of privileges to be assigned to a principal, from a variety of authorities;
- The (average) lifetimes of ACs are substantially different than those of PKCs;
- Privileges are valid only during specific intervals of time which are asynchronous with those relative to other privileges or principal's PKCs [3].
- Privileges are assigned by an issuer in a specific host. If this host belongs to a network, then such privileges are valid all over that network.

2.4 Certificate Management Models

Certificate Management Models are of two kinds: **privilege management** models and **certificate distribution** models. The first kind is specific for ACs, whereas the second kind relates to both PKCs and ACs.

[3] "The time specification extension can be used by an AA to restrict the specific period of time during which the privilege, assigned in the certificate containing this extension, can be asserted by the privilege holder" [2, 15.1.2.1].

The standard specifies four different privilege management models, as follows.

The **General model** consists of three entities: the **object**, the **privilege asserter** and the **privilege verifier**. The object represents a protected resource. The privilege asserter represents the entity that holds a set of privileges and asserts them for a particular context of use of the object. Finally, the privilege verifier is the entity that determines if the privileges asserted by the privilege asserter are sufficient for the given context of use of the object [2].

In the **Control model** there are five components (see Figure 2): the privilege asserter, the privilege verifier, the **object method**, the **privilege policy**, and the **environmental variables**. The privilege verifier controls the object method of the privilege asserter in accordance with the privilege policy and the environmental variables [2].

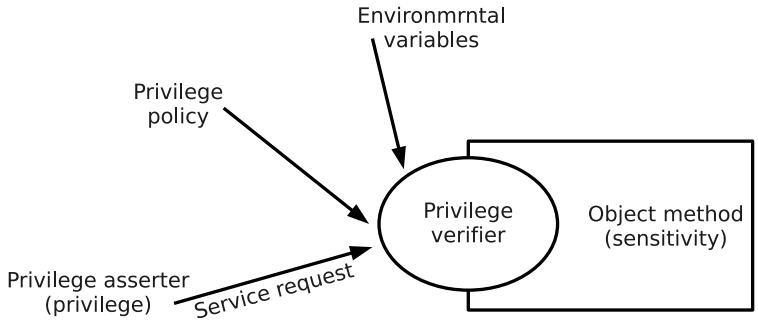

Figure 2: PMI Control Model

In the **Delegation model** there are four components (see Figure 3): the privilege verifier, the SOA, and two privilege asserters. In this case, the SOA can authorize one privilege asserter to act as AA and delegate some privileges to the other asserter, which acts as end entity. A restriction in this model is that AAs cannot delegate more privileges than they hold [2].

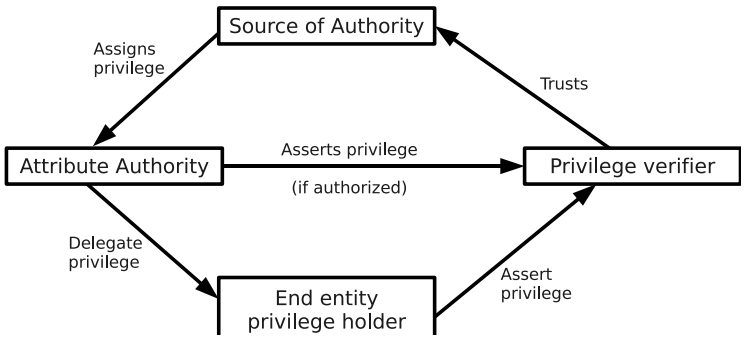

Figure 3: PMI Delegation Model

In the **Role model** users are issued **role assignment certificates** that assign one or more roles to them through the role attribute contained in the AC. Roles provide a means to indirecly assign privileges to users. In this case, it is possible to change user's privileges

without modifying user's AC, but only changing the role assignment certificate [2]. The Role model is illustrated in figure 4.

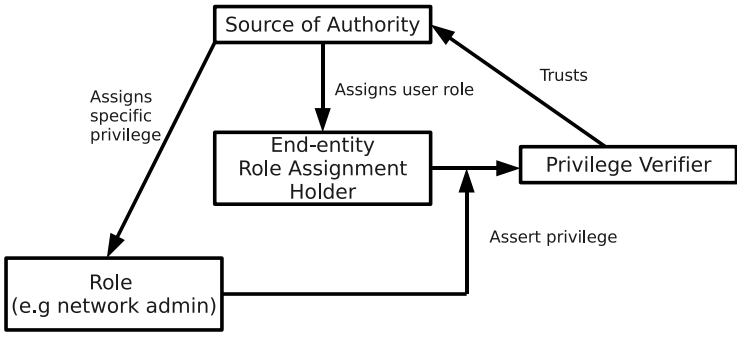

Figure 4: PMI Role Model

The distribution of digital certificates can be performed following two models: the **push** and the **pull** model. Both models assume that a principal, acting as a client, makes a request to a server in order to perform an authentication or authorization task.

In the push model, when a principal makes a request, it presents (*pushes*) a certificate to the server to corroborate the request. The server does not have to perform any search operation, but it must check that the principal's certificate is valid by inspecting a suitable certificate revocation list (see 2.3).

In the pull model, conversely, in response to the request of accessing a resource by a principal, the server requests (*pulls*) the principal's certificate from a certificate issuer or a repository [3]. Of course, the pull model simplifies the operations of the principal at the expense of those for the server.

For the case of ACs, the pull and the push model result in the two following ways to acquire a privilege by a principal:

- An AA may create an AC for the principal, without any principal's request. In this case the AC may be stored in a public repository and the AC may subsequently recovered (pulled) by one ore more **privilege verifier** to make an authorization decision.
- A principal may request a privilege to some AA. The AC, once created, may be returned (only) to the principal, which explicity supplies it when requesting access to some protected resource.

3 Naming services through LDAP

A directory is "a collection of open systems cooperating to provide directory services"[6]. A directory user, through a client, can access to a directory using a directory access protocol, and **Lightweight Directory Access Protocol** (**LDAP**) represents the *de–facto* standard directory access protocol. The main application of LDAP in the last years has been the enforcement of naming service in the context of enterprise-level distributed environments. And, indeed, directory services can be considered a specialized form of naming services,

designed to support larger and more complex information about network entities. LDAP schemas and information trees have been introduced to match the functionalities offered by other, older naming services such as DNS, NIS and NIS+.

Since in our approach we adopt LDAP for the management of both principal's authentication and authorization information, it is useful to briefly recall here some basics about LDAP (Section 3.1), and how it has been used to manage legacy OS accounting information (Section 3.2). Moreover, in Section 3.3, we will discuss the LDAP scheme recently introduced to support (Open)Solaris extended security attributes. That is a main concern for our approach, since the DD access control model relies upon these extended attributes.

3.1 LDAP data structures and operations

LDAP directories provide a way to name, manage, and access collections of directory entries. A directory entry is composed of **object classes**, which in turn compose of **attributes**, that have a **type** and one or more **values**. The syntax for each attribute defines the values allowed and how those values are interpreted during a directory operation. Directory entries are organized into a tree structure called **Directory Information Tree** (**DIT**). DITs can be based on geographic regions (country names, states, etc.), organizational boundaries (organization name and organizational units), or name service domains. Entries are named according to their position in this tree structure by a **distinguished name** (**DN**). Each component of the distinguished name is called a **relative distinguished name** (**RDN**). An RDN is composed of one or more attributes from the entry.

An LDAP **schema** is nothing more than a convenient packaging unit containing broadly similar object classes and attributes. Every attribute or object class used in an LDAP implementation must be defined in a schema, and that schema must be known to the LDAP server.

LDAP defines nine operations, categorized in the following groups:

- **Interrogation:** This group of operations interrogates the directory, retrieving its information. The operations included in this group are: `search` and `compare`.
- **Update:** This group of operations updates the directory information. The operations included in this group are: `add`, `delete`, `modify` and `modify RDN`.
- **Authentication:** The operations included in this group are: `bind`, `unbind` and `abandon`. The first and second operation provide means for securing directory information, the third operation allows users to cancel an operation in progress.

3.2 Using LDAP for legacy OS accounting information

To manage UNIX-like basic accounting information, LDAP provides the **nis.schema** [7]. The object classes for this shema are:

- `posixAccount`, for the information usually stored in `/etc/passwd`;
- `shadowAccount`, for the *shadow password*'s extensions, usually stored in `/etc/shadow`;
- `posixGroup` for information on group accounts, usually provided through the `/etc/group` file;

Figures 5, 6 and 7 show the relationships between LDAP directory entries and the above accounting information repositories.

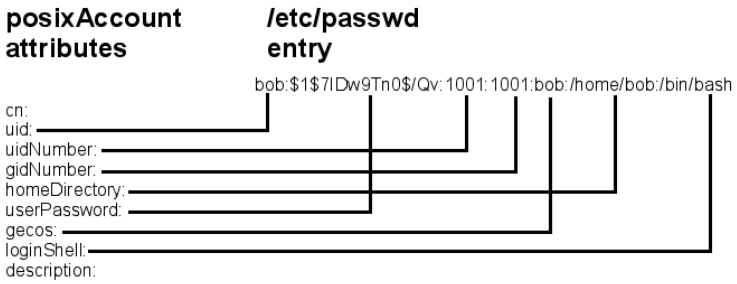

Figure 5: Relationship between posixAccount objectClass and /etc/passwd

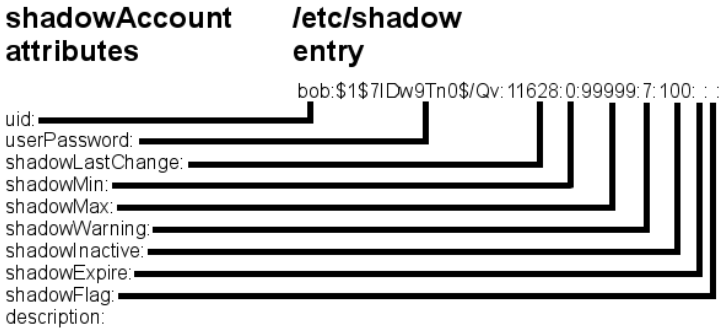

Figure 6: Relationship between shadowAccount objectClass and /etc/shadow

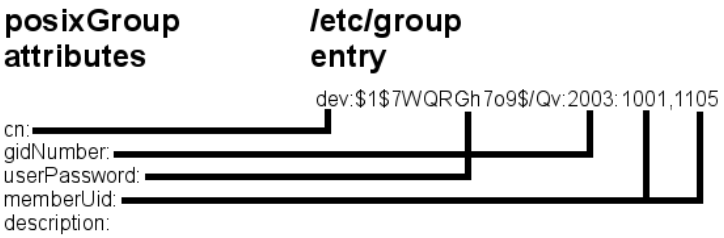

Figure 7: Relationship between posixGroup objectClass and /etc/group

The **MigrationTools**[4] are a set of Perl scripts for migrating users, groups, aliases, hosts, netgroups, networks, protocols, RPCs, and services from existing nameservices (flat files, NIS, and NetInfo) to LDAP.
These tools provide a **LDIF**[5] file for each existing system information file.
LDAP provides for special entries which act as containers to organize LDAP entries into a tree structure. Following a common practice, the containers named **People** and **Group** are used for organizing user and group accounting information, respectively. The resulting Directory Information Tree is shown in Figure 8. "Normally, all users defined in a central

[4] `http://www.padl.com/OSS/MigrationTools.html`
[5] LDAP Data Interchange Format [8]

LDAP directory have access to every host which authenticates against that directory. In some cases, it is desirable to restrict access to specific hosts for certain users defined in LDAP. This can be accomplished using the host attribute of the account objectClass." [9] The DIT for this case is shown in Figure 9

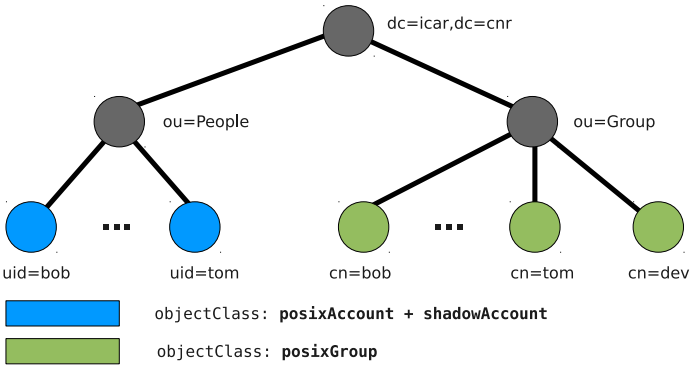

Figure 8: Directory Information Tree for UNIX-like accounting information

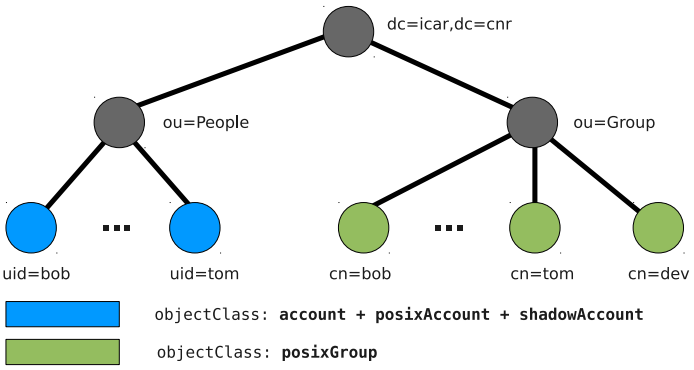

Figure 9: Directory Information Tree for UNIX-like accounting information (restricted host access)

It can be worthwhile to notice here that the management of user and group related information in a LDAP repository cannot be performed through the usual OS commands for user administration. Instead, a specific LDAP client must be used [6]

3.3 Managing Open(Solaris) security attributes with LDAP

In order to support its advanced access control policies, (Open)Solaris provides other files besides the POSIX compliant repositories for user related information. In particular, the

[6] e.g, the **Change Password Utility** (**CPU**) (`http://sourceforge.net/projects/cpu/`).

`/etc/user_attr` file is a local source of extended capabilities associated with users and roles; it lists the accounts which have addictional security attributes [7]. Each record in `/etc/user_attr` has the following structure:

```
user:qualifier:res1:res2:attr
```

where `user` is the user name as in `/etc/password`, the fields `qualifier`, `res1` and `res2` are reserved for future use, and `attr` is an optional list of semicolon-separated, key-value pairs describing the security attributes of `user`.

At the moment, (Open)Solaris interprets twelve attribute keywords, including `auths` (i.e., authorizations), `roles` and `profiles`. In turn, the definitions for these keywords are stored and managed through suitable local databases. The most relevant security attributes w.r.t. the DD access control model and DS2OS environments are discussed in some detail in Deliverable B; what is relevant in the context of this section is that the local databases specifying such advanced security attributes can be used with other user attributes sources, including the LDAP people container, the `user_attr` NIS map, and the `user_attr` NIS+ table.

LDAP schemas and DITs for (Open)Solaris are defined in [10]. The schemas specifically required for managing user related information in (Open)Solaris are the *Extended user accounting* schema, *Role based access control* schema, *Solaris client naming profile* schema, and *Projects* schema. Although the attributes stored in `/etc/user_attr` concur to support various kinds of access control policies, both of discretionary and mandatory type, they are usually referred as role based access control (RBAC) attributes, since RBAC policy is the higher level policy supported in (Open)Solaris. Thus, such attributes are described in the Role based access control schema. The role based access control Attributes are:

```
( 1.3.6.1.4.1.42.2.27.5.1.4 NAME SolarisAttrKeyValue
DESC Semi-colon separated key=value pairs of attributes
EQUALITY caseIgnoreIA5Match
SUBSTRINGS caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.7 NAME SolarisAttrShortDesc
DESC Short description about an entry, used by GUIs
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.8 NAME SolarisAttrLongDesc
DESC Detail description about an entry
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.9 NAME SolarisKernelSecurityPolicy
DESC Solaris kernel security policy
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.10 NAME SolarisProfileType
DESC Type of object defined in profile
```

[7] Be aware of not confusing the term "attribute" used in the context of Solaris security with the same term previously introduced for LDAP.

```
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.11 NAME SolarisProfileId
DESC Identifier of object defined in profile
EQUALITY caseExactIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.12 NAME SolarisUserQualifier
DESC Per-user login attributes
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.13 NAME SolarisReserved1
DESC Reserved for future use
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.14 NAME SolarisReserved2
DESC Reserved for future use
EQUALITY caseIgnoreIA5Match
SYNTAX IA5String SINGLE-VALUE )
```

The role based access control Objectclasses are instead:

```
( 1.3.6.1.4.1.42.2.27.5.2.3 NAME SolarisUserAttr SUP top AUXILIARY
DESC User attributes
MAY ( SolarisUserQualifier $ SolarisAttrReserved1 $ \
SolarisAttrReserved2 $ SolarisAttrKeyValue ) )

( 1.3.6.1.4.1.42.2.27.5.2.4 NAME SolarisAuthAttr SUP top STRUCTURAL
DESC Authorizations data
MUST cn
MAY ( SolarisAttrReserved1 $ SolarisAttrReserved2 $ \
SolarisAttrShortDesc $ SolarisAttrLongDesc $ \
SolarisAttrKeyValue ) )

( 1.3.6.1.4.1.42.2.27.5.2.5 NAME SolarisProfAttr SUP top STRUCTURAL
DESC Profiles data
MUST cn
MAY ( SolarisAttrReserved1 $ SolarisAttrReserved2 $ \
SolarisAttrLongDesc $ SolarisAttrKeyValue ) )

( 1.3.6.1.4.1.42.2.27.5.2.6 NAME SolarisExecAttr SUP top AUXILIARY
DESC Profiles execution attributes
MAY ( SolarisKernelSecurityPolicy $ SolarisProfileType $ \
SolarisAttrReserved1 $ SolarisAttrReserved2 $ \
SolarisProfileId $ SolarisAttrKeyValue ) )
```

Figure 10 illustrates the relationship between an entry of `/etc/user_attr` file and `SolarisAuthAttr` LDAP Objectclasses for the case of the example record:

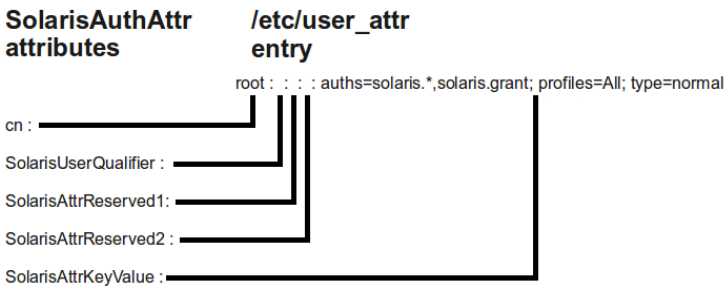

Figure 10: Reletionship between `SolarisAuthAttr` objectClass and (Open)Solaris `/etc/user_attr` file.

```
root::::auths=solaris.*,solaris.grant;profiles=All;type=normal
```

4 Implementing dynamic delegation

This part is devoted to the specifications of X.509 certificates, LDAP structures and directory service workflows for the enforcement of dynamic delegation (DD) in a DS2OS environment. In order to match the authorization information managed at the OS layer with that stored through ACs, a DD specific Attribute Certificate is required; that will be discussed in Section 4.1. Section 4.2, instead, describes the object classes and DITs that we have chosen to manage the DD model through LDAP. Section 4.3 discusses the problem of managing a principal's security context. A crucial point in such respect is the integration of the authorization information stored in X.509 certificates with that defined at the OS layer through platform specific repositories. As already explained in Deliverable A, we follow an approach which resembles the way information is accessed by naming services, requiring a suitable, incremental retrieval of principal's security attributes. Section 4.4 discusses the issue of retrieving PKCs and (related) ACs in OpenLDAP [1]. As we shall see, at the moment OpenLDAP supports only one kind of certificate's search, which does not fully accomplish the requirements for DD; we overcome these difficulties thanks to the structure of the DITs introduced in Section 4.2. Section 4.5 describes the operations for the creation, modification and deletion of a guest's account. Lastly, a brief illustration of the access control mechanisms provided for LDAP repositories is given in Section 4.6.

4.1 X.509 certificates for dynamic delegation

W.r.t. a given host H, principals in the DD access control model can be categorized as one of the following types (see Deliverable A):

- **Administrator:** administrators are granted the authorizations for managing and monitoring all other kind of principals on H. In particular, administrations are in charge of creating user accounts on H, giving them suitable authorization profiles;
- **Sponsor:** a sponsor is a principal that has an account on H and has been granted the authorization to allow principals that are not registered users on H to access and use at some extent its own resources on H.

- **Guest:** a guest is a principal that access to some resources of H thanks to, and under the constraints defined by, a sponsor in H. A guest can have more than one sponsor in H.

- **Standard User**: standard users are just principals which have an account on H and do not belong to any of the three previous categories of users.

We can relate DD principals to PMI's entities (see Section 2.3) as follows: *administrators* act as **SOA**; *sponsors* act as **AA**; *guests* and *ordinary users* act as **AC holders**.

The X.509 standard does not give definitions of any specific attribute: attribute specification does not pertain indeed to the standard, but it is environment or application specific. In order to support DD through PMIs, we introduced as LDAP attribute an **UTF8String** encoded value representing an entry of `/etc/user_attr` (see 3.3).

That attribute is conveyed using an AC which has the fields *holder* and *issuer* structured as depicted in Figures 11 and 12 (see also Sections 2.2 and 2.1). The fields marked with

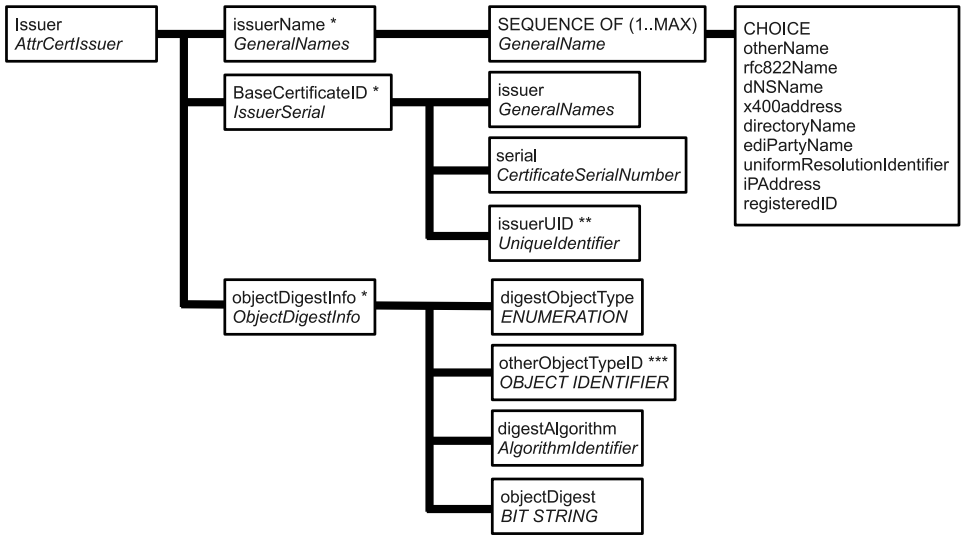

Figure 11: Structure of Issuer field.

* are optional, but at least one of them shall be present. The field marked with ** is not optional if issuerUID is present in the PKC identified by the the DN and the serial number. Finally, the field marked with *** is optional.

As respect to the linking of the above AC to its related PKC, we have preferred to adhere to the X.509 specifications instead of following the recommendation [3], since this is in conflict with the matching rule [2][17.3.1]. Using the X.509 approach, an AC can be seen as a US visa, and the PKC as an Italian passport. The US visa has the number of passport; thus, if the passport expires, or it is revoked, then the visa is invalid, too. That is coherent with the DD authorization model.

Figure 13 shows the connection between an AC, the holder's PKC and the issuer's PKC. The case depicted therein is that in which Alice acts as guest, and Bob as sponsor.

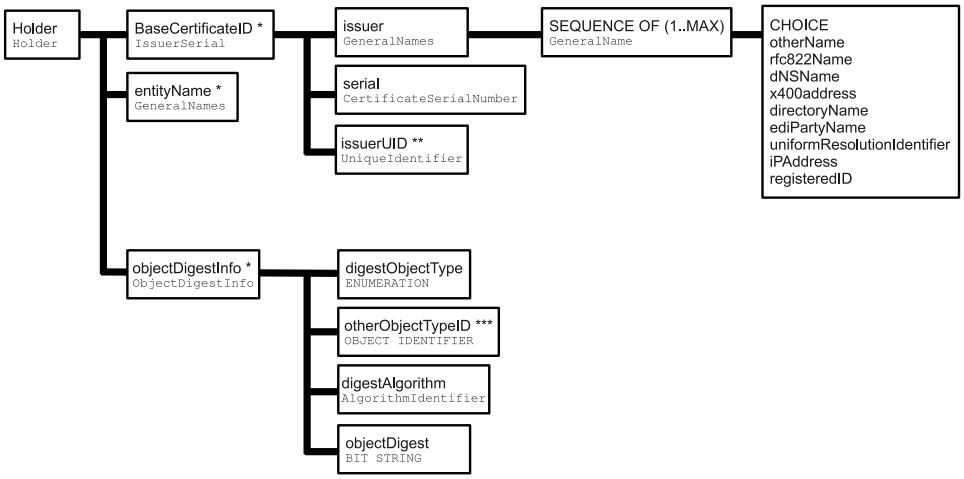

Figure 12: Structure of Holder field.

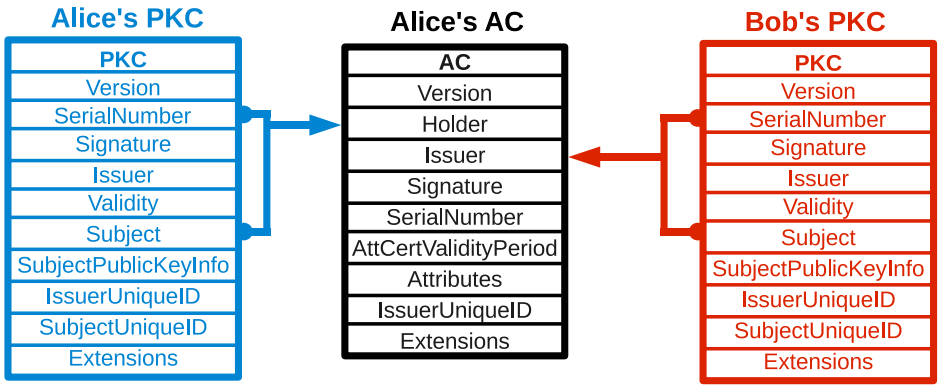

Figure 13: Connection between holder's PKC, AC and issuer's PKC

4.2 Dynamic delegation DITs

Dynamic delegation requires special Directory Information Trees or DITs (see Section 3.1). These different DITs are introduced to accomplish various organization scenarios, as we are going to show in the following, but in any case they make use of the container **People** to store user account information, the container **Group** to store group account information, and the container **Guest** to store guest information. Let us examine these three containers. **People** has two other containers as children; they are: **Sponsor** and **User**; one for each kind of DD principal, as discussed in Section 4.1.

Each account of type sponsor or ordinary user is stored in the DIT as four object classes:

1. `account:` it contains the account login name. This objectClass is defined in the

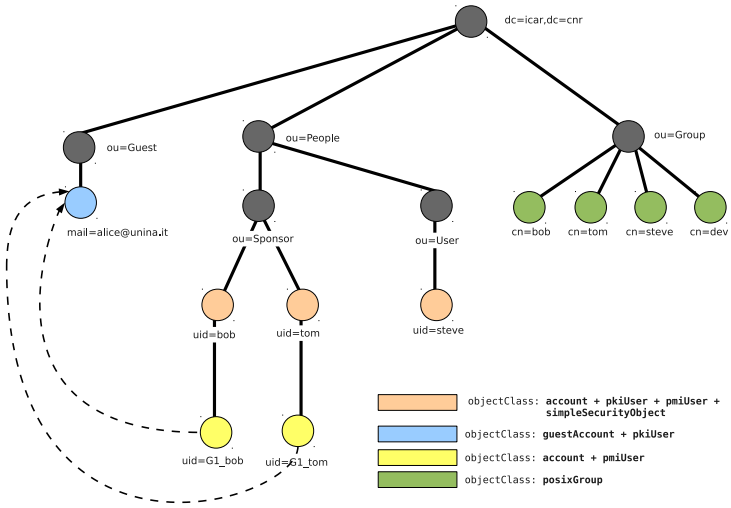

Figure 14: Directory Information Tree for dynamic delegation

`cosine.schema` [11].

2. `pkiUser:` it contains the principal's PKC, where the PKC is issued by the system administrator. This objectClass is defined in [12].

3. `pmiUser:` it contains the principal's AC, where the AC is issued by the system administrator. This objectClass is defined in [2].

4. `simpleSecurityObject:` it contains the userPassword attribute. This objectClass is defined in [11].

Since a principal can act on a given system host as a guest for more than one sponsor, then its related profile information is stored in two places in the DIT, as follows. When a new *guest* account is created, if it is the first such account for that principal, then it is stored as a child of **Guest** container, which is composed of the two `objectClass` items:

1. `guestAccount:` it composes of the username in e–mail format (attribute name is `mail`), an integer (attribute name is `count`) and one dn (attribute name is `guestuid`) for each sponsor;

2. `pkiUser:` it contains the principal's PKC, where the PKC is now issued by an external CA.

This new resulting objectClass `guestAccount` is as follows:

```
objectIdentifier delOID 1.666
objectIdentifier delLDAP delOID:2
objectIdentifier delAttributeType delLDAP:1
objectIdentifier delObjectClass delLDAP:2

objectIdentifier uid 0.9.2342.19200300.100.1.1
```

```
objectIdentifier mail 0.9.2342.19200300.100.1.3
objectIdentifier count 1.3.6.1.4.1.1466.115.121.1.27
objectIdentifier guestuid 1.3.6.1.4.1.1466.115.121.1.12

objectclass ( delObjectClass:1 NAME 'guestAccount'
    SUP top STRUCTURAL
    DESC 'guest user in dynamic delegation'
    MUST ( mail, count, guestuid )
)
```

The choice of using an email address was since it represents one component of the PKC field `subjectName`, which uniquely individuates a principal. The attributes `count` and `guestuid` were instead introduced to accomplish the need of storing in this objectClass different uids related to a same PKC, a consequence of the possible existence of multiple sponsors for a given guest, and of the restrictions about the format for the guest uid (see below).

The remaining information, which depends on the individual *sponsor*, is stored as a child of the *sponsor*'s entry. This *guest*'s entry is formed by two **objectClass**:

1. `account:` it contains the user's uid. The guest uid is obtained by appending the *sponsor* uid to the string composed by character `G`, followed by an integer (used to distinguish among multiple guests for the same sponsor), and the underscore "_" as separator. This choice was motivated by the constraints actually imposed by POSIX on the strings representing user names [8].

2. `pmiUser:` it contains the user's AC issued by its *sponsor*.

In the DITs (see, e.g., Figure 14) we put the `Guest` container at the same hierarchical level of the `People` e `Guest` ones. That is because guests usually do not belong to the organizational unit referred by the DIT, but to different, external organizations.

4.2.1 DITs for multiple units and OS istances

To implement DD in a DS2OS environment with multiple organizational units and/or OS istances, we need to extend the DIT shown in Figure 14. For example, in order to deploy DD onto the ICAR network, which composes of three branches, we add a container for each branch (i.e. `cs`, `na` and `pa`), as shown in Figure 15.

These new containers are children of the DIT's root `dc=icar,dc=cnr`, and are of type domain component (`dc`), too. The above reflects the circumstance that in this case different branches are autonomosly managed and constitute separate security domains. Different business organizations could result in different DIT architectures as well.

One more hierarchy level, the *host* level, is appropriate in case different hosts and/or computing platforms fall in distinct security or administrative domains. This case is depicted in Figure 16. Finally, Figure 17 shows the general case of multiple, independent organizational units with multiple hosts falling in separate administrative domains.

[8] According to POSIX, a user name shoud be represented by a string of no more than eight bytes consisting of characters from the set of alphabetic characters, numeric characters, period (.), underscore (_), and hyphen (-). The first character should be alphabetic and the field should contain at least one lower case alphabetic character. A warning message is displayed if these restrictions are not met.

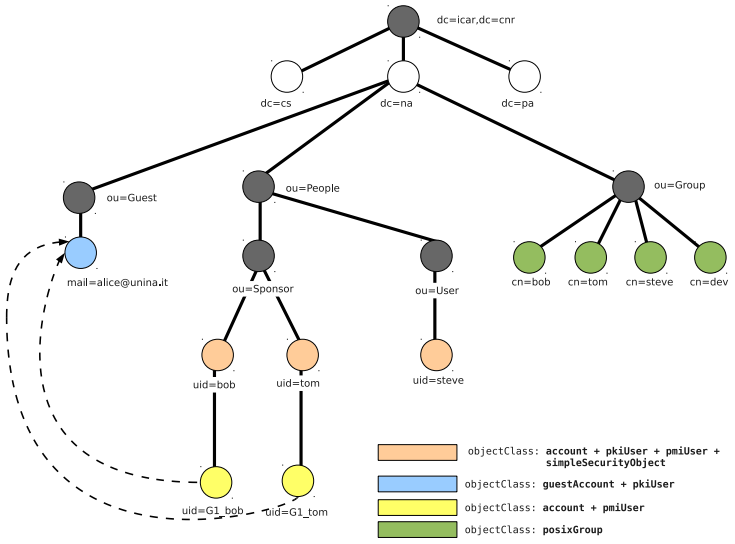

Figure 15: Directory Information Tree for DD when there are more branches

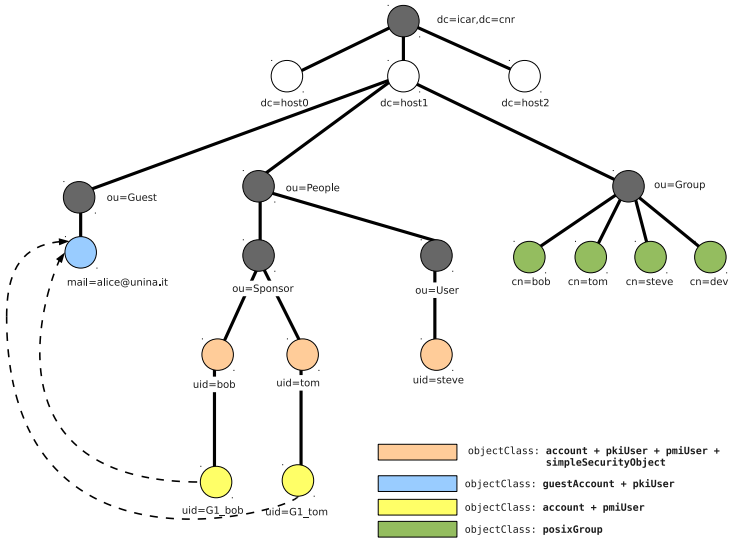

Figure 16: Directory Information Tree for DD when there are more OS istances

4.3 Security context management

This section illustrates the approach, introduced in Deliverable A, to allow for the acquisition of authorization information concerning principals in a DS2OS environment.
We remember that, according to one directory services requirement (see Deliverable A), such authorization information must be achieved thanks to the *name service switch framework*,

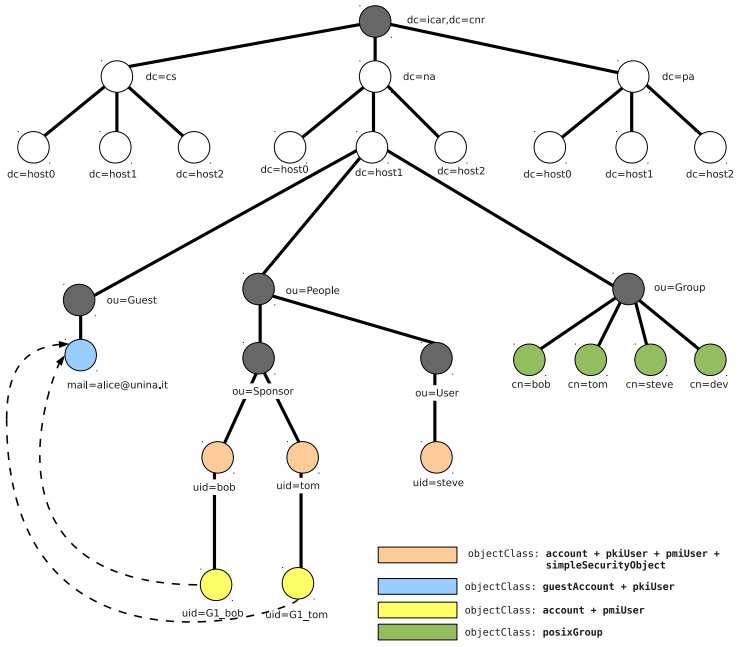

Figure 17: Directory Information Tree for DD when there are more branches and more OS istances

in an incremental and consistent way that gives (at least by default) higher priorities to local repositories and lower priorities to remote ones. That requirement applies to all principal's profiles (see Section 4.1), and in particular to the guest and sponsor profiles. However, we have to distinguish between guest and non-guest profiles, as follows.

In case of a principal p with a non-guest profile u on host H, its set of security attributes $A_H(u)$ is first retrieved from suitable local repositories on H, and then extended using ACs stored in one or more LDAP directory service via a special *exclusive union* operation (see below). Instead, if p whishes to operate on H as the guest profile g, then $A_H(g)$ is first desumed from the credentials issued for g by its sponsor s in H (via local repositories and/or ACs stored in LDAP directory services), and then restricted at runtime via a special *exclusive intersection* operation with the security attributes set $A_H(s)$ of s (see below). This guarantees that the security attributes for g are a subset of those for s also if the system administrator removes some credential for s after its granting of credentials to g. In particular, if s has revoked its credential to act as a sponsor, then g can no more operate as guest of s on H.

Exclusive union, or shortly **x-union**, is a special set operation which applies to sets of boolean propositions only. Given two sets of security attributes A and B, $A \sqcup B$ (to be read "A x-united B") is the set of attributes which is obtained by adding to the attributes in A only those attributes in B that neither get in conflict nor duplicate the attributes in A. For example, given the three sets of attributes $A = \{a_1, a_2, a_4, !a_5\}$, $B = \{a_1, a_3, a_5\}$ and $C = \{a_2, !a_3, a_4, a_6\}$, where "!" denotes the negation operator, we have that

$$A \sqcup B \quad = \quad \{a_1, a_2, a_3, a_4, !a_5\}$$

$$(A \sqcup B) \sqcup C \quad = \quad \{a_1, a_2, a_4, !a_5, a_6\}$$

Notice that, conversely than for the standard set union operation, the commutative property does not hold for the x-union operation.

Exclusive intersection, or shortly **x-intersection**, is a special set operation which applies to sets of boolean propositions only. Given two sets of security attributes A and B, $A \sqcap B$ (to be read "A x-intersected B") is the set of attributes obtained by taking only the attributes common to A and B, with the exception that if the second set B does not contain a special attribute, then the intesection is the empty set. In our case, the special attribute is the credential a_s of *being a sponsor*. For example, given the two sets of attributes $A = \{a_1, a_2, a_4\}$, $B = \{a_1, a_3, a_5, a_s\}$, we have that

$$\begin{aligned} A \sqcap B &= \{a_1\} \\ B \sqcap A &= \emptyset \end{aligned}$$

Thus, also the x-intersection operation does not satisfy the commutative property.

4.4 Retrieval of certificates

At the present moment, OpenLDAP (`http://www.openldap.org`) is the only opensource LDAP protocol implementation that can store and retrieve both PKCs and ACs.
As for PKCs, OpenLDAP supports at the moment only the **CertificateExactAssertion** search syntax, defined in [2][11.3.1]. This kind of search allows to retrieve PKCs having specific *issuer* and *serialNumber* values. However, common tasks in DS2OS environments, such as the logging of a guest user to a remote host, require the retrieval of a certificate based on the value of its *subject* field, which is realized through the **CertificateAssertion** search syntax [2, 11.3.2].

Analogously, the current OpenLDAP implementation only (partially) supports the **AttributeCertificateExactAssertion** search syntax (defined in [2, 17.3.1]), which allow to retrieve an AC having a specific `serialNumber` and a specific `issuer` [9]. Instead, connecting a PKC holder to an AC, as required by DD, would require the implementation of the **HolderIssuerAssertion** search syntax defined in [2, 17.3.3].

The above difficulties can be overcome thanks to the DITs introduced so far. PKC and AC are indeed stored in specific DIT entries, which can be accessed through the knowledge of three kinds of data:

- the guest's username, inserted at login time by the *guest user*;
- the sponsor's username, inserted at login time by the *guest user*;
- the host OS istance, which is known by the host (only in case of multiple OS instances, as for the DITs of Figures 16 and 17).

4.5 Guest account management

This section illustrates the steps that a sponsor must perform to create, edit and delete a guest's account.
It is intended that these operations are performed through a suitable guest account management tool built upon legacy OS commands like `useradd`, `userdel`, etc., and OpenLDAP specific commands such as `ldapadd`, `ldapmodify`, `ldapsearch`, etc. We realized a first version of such a program in bash scripting language with a GUI written in Java.

[9] the field `issuer` is composed by the PKC `serialNumber` and the `subject` used to sign the AC

4.5.1 Guest account creation

To create a new guest account on host H for user G, a sponsor S through the above tool or GUI has to perform the following tasks:

1. creation of G's AC containing its own authorization attributes;
2. editing of user related repositories of H (e.g., `/etc/passwd` and `/etc/group`) to insert G's accounting information;
3. storing of the AC created at the step 1 in an LDAP network repository;
4. (optional) if the above constitutes the first guest account for G in that organizational unit, then S must create a new instance of the couple of objectClasses `guestAccount` and `pkiUser` under the container `Guest` (the blue node in the DITs presented in Section 4.2);
5. S must create a new guest entry, that is a new instance of the couple of objectClasses `account` and `pmiUser` under his own uid container. Such entries are depicted as yellow nodes in the DITs of Section 4.2.

4.5.2 Guest account upgrade

In order to modify an existing guest account, a sponsor S must create a new AC for the guest G. After the AC creation, S must update the guest entry for G under his own uid container (a yellow node in the DITs shown in Section 4.2).
If G is logged in during the account upgrade, the modification will take effect at the next login.

4.5.3 Guest account deletion

To delete a guest account for G on host H, a sponsor S must delete G's information concerning that specific account from both user related repositories on H and LDAP databases. The guest account entry composed of the two objectClasses `guestAccount` and `pkiUser` (the blue node in the DITs of Section 4.2) has to be removed just in case G has no other guest accounts in the considered organizational unit. For example, referring to Figure 14, deletion of the guest account `G1_bob` by her sponsor `uid=bob` consists just in the removal of the related yellow node in the DIT, whilst the successive deletion of `G1_tom` results in the removal of the yelloe node and the blue node `uid=alice@unina.it` under the ou Guest.

4.6 Security considerations

LDAP provides an **Access Control List** (**ACL**) mechanism which can be enforced to protect specific attributes or entire subtrees of a DIT. The following example shows an ACL for `userPassword` attribute.

```
# ACL1 for userPassword attribute
access to attrs=userPassword
    by self      write
    by anonymous auth
    by "cn=Manager,dc=icar,dc=cnr" write
    by *         none
```

Thanks to the LDAP ACL mechanism, it is easy to constrain the guest account management tool to operate just on the information that pertain to a given sponsor S. In order to allow for that, S must have a DIT entry with a `userPassword` attribute for his authentication to the LDAP repository.
The following examples show two such ACLs. The first ACL allows any sponsor to create and modify the **ou=Guest** subtree. The second one, allows a given sponsor S to create and modify his entry subtree only.

```
# ACL2 for ou=Guest,dc=icar,dc=cnr subtree
access to dn.subtree="ou=Guest,dc=icar,dc=cnr"
    by self     read
    by anonymous read
    by "cn=Manager,dc=icar,dc=cnr" write
    by dn.one="ou=Sponsor,ou=People,dc=icar,dc=cnr" write
    by "cn=PrivilageVerifier,dc=icar,dc=cnr" read
    by * read


# ACL3 for subtree
# uid=<sponsor_username>,ou=Sponsor,ou=People,dc=icar,dc=cnr
access to dn.regex="(.+,)?(uid=[^,]+,ou=Sponsor,ou=People,dc=icar,dc=cnr)$"
    by dn.exact,expand="$2" write
    by "cn=Manager,dc=icar,dc=cnr" write
    by "cn=PrivilageVerifier,dc=icar,dc=cnr" read
    by * none
```

References

[1] OpenLDAP Foundation. *OpenLDAP Admin Guide (version 2.4)*. OpenLDAP Foundation, January 2009. `http://www.openldap.org/doc/admin24`.

[2] International Telecommunication Union Telecommunication Standardization Sector. *The Directory: Public-key and attribute certificate frameworks*. ITU-T, August 2005.

[3] R. Housley S. Farrell and S. Turner. *RFC 5755: An Internet Attribute Certificate Profile for Authorization*. RFC Editor, January 2010.

[4] M. Toorani and A.A.B. Shirazi. Lpki - a lightweight public key infrastructure for the mobile environments. In *11th IEEE International Conference on Communication Systems (IEEE ICCS'08) - (Guangzhou, China, November 2008)*, pages 162 – 166. IEEE, 2008.

[5] D.W. Chadwick. An X.509 role based privilege management infrastructure. In *Briefing - Global InfoSecurity 2002, World Markets Research Centre Ltd*. World Markets Research Centre, October 2001. On accompanying CD-ROM Reference Library/03.pdf.

[6] International Telecommunication Union Telecommunication Standardization Sector. *The Directory: Overview of Concepts, Models and Service*. ITU-T, November 1993.

[7] L. Howard. *RFC 2307: An Approach for Using LDAP as a Network Information Service*. RFC Editor, March 1998.

[8] G. Good. *RFC 2849: The LDAP Data Interchange Format (LDIF) - Technical Specification*. RFC Editor, June 2000.

[9] M. Gnirss and F. Kirschner. *Advanced LDAP User Authentication: Limiting Access to Linux Systems Using the Host Attribute*. IBM Redbooks, April 2004.

[10] Inc. Sun Microsystem. *LDAP Setup and Configuration Guide*. Sun Microsystem, Inc., January 2001.

[11] K. Zeilenga. *RFC 4524: COSINE LDAP/X.500 Schema*. RFC Editor, June 2006.

[12] K. Zeilenga. *RFC 4523: Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates*. RFC Editor, June 2006.