CAS++: An Open Source Single Sign-On Solution for Secure e-Services

Claudio Agostino Ardagna, Ernesto Damiani,
Sabrina De Capitani di Vimercati, Fulvio Frati, and Pierangela Samarati

Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
Via Bramante 65 - Crema - Italy
{ardagna,damiani,decapita,frati,samarati}@dti.unimi.it

Abstract. Business and recreational activities on the global communication infrastructure are increasingly based on the use of remote resources and services, and on the interaction between different, remotely located parties. On corporate networks as well as on the open Web, the huge number of resources and services often requires to multiple log-ons leading to credential proliferation and, potentially, to security leaks. An increasingly widespread approach to simplify and secure the log-on process is Single Sign-On (SSO) that allows automatic access to secondary domains through a single log-on operation to a primary domain. In this paper, we describe the basic concepts of SSO architecture focusing on the central role of open source implementations. We outline three major SSO trust models and the different requirements to be addressed. We then illustrate CAS++, our open source implementation of a Single Sign-On service. Finally, we illustrate the application of CAS++ to a real case study concerning the development of a multi-service network management system. The motivation for our work has been raised in response to the requirements of such case study within the Pitagora project.

1 Introduction

Applications running on the Global Information Infrastructure are increasingly designed by composing individual *e-services* such as e-Government services, remote banking, and airline reservation systems [12], providing various kind of functionalities such as paying fines, renting a car, releasing authorizations, and so on. From the architectural point of view, service-oriented distributed applications follow a layered software structure composed of three layers [16]: *i) e-Service components,* software components implementing e-services; *ii) Application server*, a middleware layer over which the components will be deployed and that provides some additional functionalities such as management of security and persistence; *iii) Operating System platform*, over which the application will be distributed. While there is an increasing need for authenticating clients of these applications before granting them access to services and resources, individual e-services are rarely designed in such a way to handle the authentication process. The reason e-services do not include functionalities for checking the client's credentials is that they assume a unified directory system to be present, making

Please use the following format when citing this chapter:
Author(s) [insert Last name, First-name initial(s)], 2006, in IFIP International Federation for Information Processing, Volume 201, Security and Privacy in Dynamic Environments, eds. Fischer-Hubner, S., Rannenberg, K., Yngstrom, L., Lindskog, S., (Boston: Springer), pp. [insert page numbers].

suitable authentication interfaces available to client components of network applications. On some corporate networks, all users have a single identity across all services and all applications are directory enabled. As a result, users only log in once to the network, and all applications across the network are able to check their unified identities and credentials when granting access to their services. However, on most Intranet and on the open network users have multiple identities, and a solution is needed to give them the illusion of having a single identity and a single set of credentials. Single Sign-On (SSO) systems are aimed at providing this functionality, managing the multiple identities of each user and presenting their credentials to network applications for authentication. In this paper, we describe a fully functional *open source* Single Sign-On [7] solution, that allows users to enter a single username and password to access systems and resources, to be used in the framework of an open source e-service scenario. Indeed, open specifications for inter-organizational SSO do exist; for example, the Liberty Alliance (LA) project, started in 2001 and involving more than 130 organizations, is aimed at providing a framework for protecting business transaction, and its scope clearly includes open standards for *federated network identity*. However, here we shall focus on specific open source implementations of SSO systems, which may or may not fully comply to open specifications like LA. As a matter of fact, in many application fields open source products are increasingly being adopted as an alternative to proprietary solutions. In particular, our work has been driven by the requirements for an open source Single Sign-On solution raised within the Pitagora project, where we are collaborating with Siemens Mobile for the development of a multi-service network management system.

2 Single Sign-On: Basic Concepts

The huge amount of services available on the Net is causing a proliferation of user accounts. Users typically have to log-on to multiple systems, each of which may require different usernames and authentication information. All these accounts may be managed independently by local administrators within each individual system [20].

In a multiservice domain, each system acts as an independent domain. The user first interacts with a *primary domain* to establish a session with that domain. This transaction requires the user to provide a set of credentials applicable to the primary domain. The primary domain session is usually represented by an operating system shell executed on the user's workstation. From this primary domain session shell, the user can require services from other *secondary domains*. For each of such requests the user has to provide another set of credentials applicable to the interested secondary domain.

From the account management point of view, this approach requires independent management of accounts in each domain and use of different authentication mechanisms. In the course of time, several usability and security concerns have been raised leading to a rethinking of the log-on process aimed at co-ordinating and, where possible, integrating user log-on mechanisms of the different domains.

A service/architecture that provides such a co-ordination and integration is called *Single Sign-On* [13]. In the SSO approach the primary domain is responsible for collecting and managing all user credentials and information used during the authentication

process, both to the primary domain and to each of the secondary domains that the user may potentially require to interact with. This information is then used by Single Sign-On services within the primary domain to support the transparent authentication by each of the secondary domains with which the user requests to interact. The advantages of the SSO approach include [11, 20]:

- *reduction* of *i)* the *time spent* by the users during log-on operations to individual domains, *ii) failed log-on* transactions, *iii)* the *time used to log-on* to secondary domains, *iv) costs and time* used for users profiles administration;
- *improvement to users security* since the number of username/password each user has to manage is reduced;[1]
- *secure and simplified administration* because with a centralized administration point, system administrators reduce the time spent to add and remove users or modify their rights;
- *improved system security* through the enhanced ability of system administrators to maintain the integrity of user account configuration including the ability to change an individual user's access to all system resources in a co-ordinated and consistent manner;
- *improvement to services usability* because the user has to interact with the same login interface.

SSO provides a uniform interface to user accounts management thus enabling a coordinated and synchronized management of the component domains.

3 Trust Models and Requirements of Single Sign-On Solutions

The definition of different trust models is important for the evaluation of different SSO solutions, which could slightly differ in their purposes depending on the business and trust scenario in which they act. For the goal of our analysis, we define three trust models over which the requirements, defined in Section 3.2, will be categorized.

3.1 Trust Models

A trust model describes a system through the definition of the underlying environment and of its behaviors, components, and rules. In particular, the model defines the entities involved in the system, the rules that regulate the interactions between the entities and the peculiarities of the overall system. Trust models are the basis for interoperability. For our goals, we focus on the definition of trust models in SSO environments based on the services that these environments support. We have identified three models.

Authentication and Authorization Model (AAM). This model represents one of the traditional security/trust models describing all the frameworks that provide authentication and authorization features [10]. It represents the basic mechanism in which

[1] It is important to note that, while improving security since the user has less accounts to manage, SSO solutions imply also a greater exposure from attacks; an attacker getting hold of a single credential can in principle compromise the whole system.

a user requires an access to a service that checks the users' credentials to decide whether access should be granted or denied. This model identifies two major entities: *users,* which request accesses to resources, and *services,* potentially composed by a set of intra-domain services, which share these resources. This model is based on the classic client-server architecture and provides a generic protocol for authentication and authorization processes.

Federated Model (FM). This model represents one of the emergent security/trust models in which several homogeneous entities interact to provide security services, such as identity privacy and authentication. This model identifies two major entities: *users,* which request accesses to resources, and *services,* which share these resources. The major difference with the previous model resides in the service definition and composition: in federated systems the services are distributed on different domains and they are built on the same level allowing mutual trust and providing functionalities as cross-authentication [17].

Full Identity Management Model (FIMM). This model represents one of the most challenging security and privacy/trust models that, potentially, could merge the previous two models. In addition, it provides mechanisms for identity and account management and privacy protection [3, 18]. This model identifies three major entities: *users,* which request accesses to resources, *services,* which share these resources, and *identity manager,* which gives functionalities to manage users identities. The major difference with the previous models is that FIMM tries also to fulfill the needs of privacy that arise in emerging scenarios.

3.2 Requirements

The requirements that a Single Sign-On solution should satisfy are more or less well known within the security community, and several SSO projects published partial lists. [2] However, to the best of our knowledge no complete discussion of high-level functional requirements for SSO has been published yet. A first step before implementing an open source innovative SSO system consists in spelling out these requirements, taking lessons learned from previous projects into account. Our analysis brought us to formulating the following requirements (for each requirement we report the trust model (AMM, FM, FIMM) to which it refers).[3]

Authentication (AAM,FM,FIMM). The main feature of a SSO system is to provide an authentication mechanism. Usually the authentication is performed through the classic username/password log-in, whereby a user can be unambiguously identified. Authentication mechanisms should usually be coupled with a logging and auditing process to prevent and, eventually, find out malicious attacks and unexpected behaviors. From a software engineering point of view, authentication is the only "necessary and sufficient" functional requisite for a SSO architecture.

[2] For an early attempt at a SSO requirements list, see `middleware.internet2.edu/webiso/docs/draft-internet2-webiso-requirements-07.html`

[3] Note that, different models fulfill a different set of requirements (see Table 3.2). SSO solution should be evaluated therefore by taking into consideration only the requirements supported by the corresponding trust model.

Table 1. Requirements categorization basing on the specific trust model.

Requirement	AAM Model	FM Model	FIMM Model
Authentication	X	X	X
Strong Authentication	X	X	X
Authorization	X		X
Provisioning	X		X
Federation		X	X
C.I.M. (Centralized Identity Management)	X		X
Client Status Info	X	X	X
Single Point of Control	X		
Standard Compliance	X	X	X
Cross-Language availability	X	X	X
Password Proliferation Prevention	X	X	X
Scalability	X	X	X

Strong Authentication (AAM,FM,FIMM). For high security environments, the traditional username/password authentication mechanism is not enough. Malicious users can steal a password and act in place of the user. New approaches are therefore required to better protect services against unauthorized accesses. A good solution to this problem could integrate username/password with strong authentication mechanism based on two-factor authentication such as a smartcard and biometric properties of the user (fingerprints, retina scan, and so on).

Authorization (AAM,FIMM). After the authentication process, the system can determine the level of information/services the requestor can see/use. While application based on domain specific authorizations can be defined and managed locally at each system, the SSO system can provide support for managing authorizations (e.g. role or profile acquisitions) that apply to multiple domains.

Provisioning (AAM,FIMM). Provisions are those conditions that need to be satisfied or actions that must be performed before a decision is taken [6]. A provision is as a pre-condition; it is responsibility of the user to ensure that a request is sent in an environment satisfying all the pre-conditions. The non-satisfaction of a provision implies a request to the user to perform some actions.

Federation (FM,FIMM). The concept of *federation* is strictly related to the concept of *trust*. A user should be able to select the services that she wants to federate and defederate to protect her privacy and to select the services to which she will disclose her own authorization assertions.

C.I.M. (Centralized Identity Management) (AAM,FIMM). The centralization of authentication and authorization mechanisms and, more generally, the centralization of identity management implies a simplification of the user profile management task. User profiles should be maintained within the SSO server thus removing such a burden from local administrators. This allows a reduction of user-profile administration cost and time and improves administrators' control on user profiles and authorization policies.

Client Status Info (AAM,FM,FIMM). The SSO system architecture implies the exchange of user information between SSO server and services to fulfill authentication and authorization processes. In particular, when the two entities communicate, they have to be synchronized on what concern the user identity; privacy and security issues need to be addressed. Different solutions of this problem could be adopted involving either the transport (e.g. communication can be encrypted) or the application layer.

Single Point of Control (AAM). The main objectives of a SSO implementation are to provide a unique access control point for users who want to request a service, and, for applications, to delegate some features from business components to an authentication server. This point of control should be unique to clearly separate the authentication point from business implementations, avoiding the replication and the ad-hoc implementation of authentication mechanisms for each domain. Note that every service provider will eventually develop its own authentication mechanism.

Standard Compliance (AAM,FM,FIMM). It is important for a wide range of applications to support well-known and reliable protocols to make possible communication and integration between different environments. In a SSO scenario, there are protocols for exchanging messages between authentication servers and service providers, and between technologies, within the same system, that can be different. Hence, every entity can use standard technologies (e.g. X.509, SAML for expressing and exchanging authentication information and SOAP for data transmission) to interoperate with different environments.

Cross-Language availability (AAM,FM,FIMM). The widespread adoption of the global Internet as an infrastructure for accessing services has consequently influenced the definition of different languages/technologies used to develop these applications. In this scenario, a requisite of paramount importance is the development of SSO solutions that permit the integration of service implementations based on different languages, without substantial changes to service code. The first step in this direction is the adoption of standard communication protocols based on XML.

Password Proliferation Prevention (AAM,FM,FIMM). A well-known motivation for the adoption of SSO systems is the prevention of password proliferation so to improve security and simplify user log-on actions and system profile management.

Scalability (AAM,FM,FIMM). An important requirement for SSO systems is to support and correctly manage a continuous growth of users and subdomains that rely on them, without substantial changes to system architecture.

4 Our Solution: CAS++

We have developed our open source SSO system with the goal of addressing the AAM requirements identified in the previous section by properly extending an existing open source SSO implementation, named *Central Authentication Service* (CAS) [5, 8]. In this section, we briefly describe CAS and then illustrate our solution, called *CAS++*, developed as an extension to CAS. Note that, CAS++ is not the only implementation available on the Net. In particular, *SourceID* [21], an Open Source implementation of the SSO Liberty Alliance, *Java Open Single Sign-On* (JOSSO) [15], and *Shibboleth* [19] stand out as the most complete available SSO solutions.

4.1 Central Authentication Service (CAS)

Central Authentication Service (CAS) [5, 8] is an open source framework developed by Yale University and implements a SSO mechanism to provide a *Centralized Authentication* to a single server and *HTTP redirections*. CAS authentication model is loosely based on classic Kerberos-style authentication. When an unauthenticated user sends a service request, this request is redirected from the application to the authentication server (CAS Server), and then back to the application after the user has been authenticated. The CAS Server is therefore the only entity that manages passwords to authenticate users and transmits and certifies their identities. The information is forwarded by the authentication server to the application during redirections by using session cookies (see data flow in Figure 2).

CAS is composed of pure-Java servlets running over any servlet engine and provides a very basic web-based authentication service. In particular, its major security features are:

1. passwords travel from browsers to the authentication server via an encrypted channel;
2. re-authentications are transparent to users if they accept a single cookie, called *Ticket Granting Cookie* (TGC). This cookie is opaque (i.e., TGC contains no personal information), protected (it uses SSL) and private (it is only presented to the CAS server);
3. applications know the user's identity through an opaque one-time Service Ticket (ST) created and authenticated by the CAS Server, which contains the result of a hash function applied to a randomly generated value.

Also, CAS credentials are *proxiable*. At start-up, distributed applications get a *Proxy-Granting Ticket* (PGT) from CAS When the application needs access to a resource, it uses the PGT to get a proxy ticket (PT). Then, the application sends the PT to a back-end application. The back-end application confirms the PT with CAS, and also gains information about who proxied the authentication. This mechanism allows "proxy" authentication for Web portals, letting users to authenticate securely to untrusted sites (e.g., student-run sites and third-party vendors) without supplying a password. CAS works seamlessly with existing Kerberos authentication infrastructures and can be used by nearly any Web-application development environment (JSP, Servlets, ASP, Perl, mod_perl, PHP, Python, PL/SQL, and so forth) or as a server-wide Apache module. Also, it is freely available from Yale University (with source code).

4.2 CAS++

We developed an open source SSO system, called CAS++, based on the use of identity certificates and fully integrated with the JBoss security layer. Our solution integrates the CAS system with the authentication mechanism implemented by a Public Key Infrastructure (PKI). CAS++ implements a fully multi-domain stand-alone server that provides a simple, efficient, and reliable SSO mechanism through HTTP redirections, focused on user privacy (opaque cookies) and security protection. CAS++ permits a

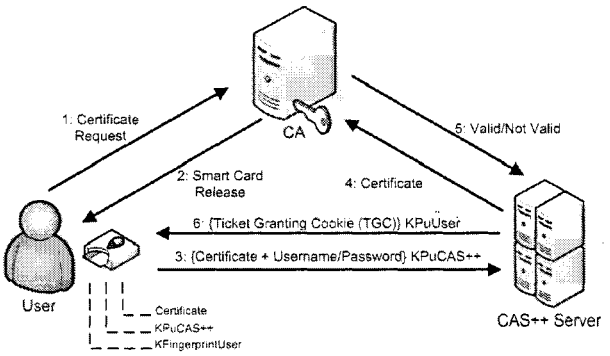

Fig. 1. CAS++ certificate-based authentication flow.

centralized management of user profiles granting access to all services in the system with a unique pair username/password. The profiles repository is stored inside the SSO server application and is the only point where users credentials/profiles are accessed, thus reducing information scattering. In our implementation, services do not need an authentication layer because this feature is managed by CAS++ itself.

CAS++ relies on standard protocols such as SSL, for secure communications between the parties, and X.509 digital certificates for credentials exchange. Besides being a "pure-Java" module like its predecessor, CAS++ is a fully J2EE compliant application integrable with services coded with every web-based implementation language. It enriches the traditional CAS authentication process through the integration of biometric identification (by fingerprints readers) and smart card technologies in addition to traditional username/password mechanism, enabling two authentication levels. Our strong authentication process flow is composed of the following steps (see Figure 1):[4]

1. the user requests an identity certificate to the CA (Certification Authority);
2. the user receives from the CA a smart card that contains a X.509 identity certificate, signed with the private key of the CA, that certifies the user identity. The corresponding user private key is encrypted with a symmetric algorithm (e.g., 3DES) and the key contained inside the smart card can be decrypted only with a key represented by user fingerprint (KFingerprintUser)[14];
3. to access a service the public key certificate, along with the pair username/password, is encrypted with the CAS++ public key (KPuCAS++) and sent to CAS++;
4. CAS++ decrypts the certificate with its private key, verifies the signature on the certificate with the CA public key, and verifies the validity of this certificate by interacting with the CA;
5. CAS++ retrieves from the CA information about the validity of the user certificate encrypted with KPuCAS++;

[4] Note that, the first two actions are performed only once when the user requests the smart card along with an identity certificate.

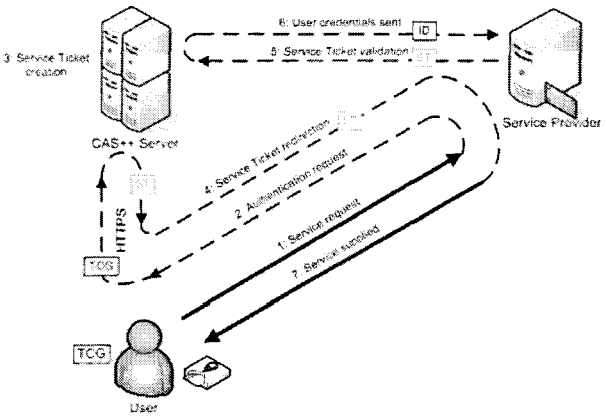

Fig. 2. CAS++ information flow for service request evaluation.

6. if the certificate is valid, CAS++ extracts the information related to the user, creates the ticket (TGC, Ticket Granting Cookie) and returns it to the user encrypted with the public key of the user (KPuUser). At this point, to decrypts the TGC, the user must retrieve the private key stored inside the smart card by mean of her fingerprint. As soon as the card is unlocked, the private key is extracted and the TGC decrypted. This ticket will be used for every further access, in the same session, to any application managed by the CAS++ Single Sign-On server.

At this point, for every further access in the session, the user can be authenticated by the service providing only the received TGC without any additional authentication action.[5]

The service access flow, that takes place over secure channels and is similar to the one in CAS, is composed of the following steps (see Figure 2):

1. the user, via a web browser, requests access to the service provider;
2. the service provider requests authentication information through a HTTP redirection to the CAS++ Server;
3. the CAS++ Server retrieves the user TGC and the service requested URL. If the user has been previously authenticated by CAS++ and has the privilege to access the service a Service Ticket is created;
4. the CAS++ Server redirects the user browser to the requested service along with the ST;
5. service receives the ST and check its validity sending it to the CAS++ Server;
6. if the ST is valid the CAS++ Server sends to the Service an XML file with User's credentials;
7. the user gains the access.

[5] Note that the TGC lifetime should be relatively short to avoid conflicts with the CA's certificate revocation process, which could cause unauthorized accesses.

Table 2. Evaluation of CAS++ with respect to the requirements of the AAM model.

Requirement	CAS++
Authentication	yes
Strong Authentication	yes
Authorization	yes
Provisioning	planned
C.I.M. (Centralized Identity Management)	yes
Client Status Info	yes (opaque)
Single Point of Control	yes
Standard Compliance	partial (HTTP, SSL, X.509)
Cross-Language Availability	yes
Password Proliferation Prevention	yes
Scalability	planned

4.3 Evaluating CAS++ Against the AAM Requirements

CAS++ is based on the Authentication and Authorization Model. Table 4.3 reports the results of the evaluation of CAS++. As it is visible from this table, CAS++ fulfills most of AAM requirements; it provides a central point of control to manage authentication, authorization, and user profiles.[6] Furthermore, CAS++ enriches the traditional CAS authentication process with the integration of biometric identification (via fingerprints readers) and smart card technologies and it is planned to include provisioning features in future releases. Note that, the lower level of CAS++ system is language independent and relies on traditional established standards, such as HTTP, SSL and X.509, without adopting emerging ones, such as SOAP and SAML. Finally, focusing on client status info, all communications between user browser, services providers and authentication server in CAS++ scenario are managed by the exchange of opaque cookies and by the use of encrypted channels.

5 A Case Study: the Pitagora Project

The increasing usage of GSM mobile phones and the upcoming of a new generation of mobile system (called third-generation or 3G) have lead to the development of applications that manage the mobile network and provide new services to users. In this scenario, every network technician that has to use multiple parallel services must manage several pairs username/password, raising all the problems discussed in the previous sections of this paper. In particular, the adoption of SSO, with strong authentication mechanisms through smart card and fingerprint readers, allows also the restriction of simultaneous multi-accesses for security reasons; in our scenario, in fact, we manage very sensitive data and, in some cases, we want to avoid any kind of data correlation.

[6] The centralization of users profiles affects system scalability. A solution that provides a balance between centralization and scalability needs is under study.

Focusing on this scenario, we show a case study example that involved our SSO implementation integrated with the research and development project "Pitagora", carried out by our group in cooperation with Siemens Mobile. Currently, the Pitagora Project is composed of the following applications:

Web-based MultiProtocol User Interface (IMW): is the application tool that provides and controls the access to OMC (Operation and Maintenance Center) system requested by users/ technicians. In particular, users are able to manage, configure, and check OMC mobile network using different technologies and devices, such as traditional PCs/laptops, PDAs, mobile phones. Hence, IMW manages all the communication process between users and OMC system, through different technologies as web browser and HTTP/HTTPS protocol, WAP browser, SMS. IMW keeps network technicians up-to-date on the network state, notifying alarms and warnings, at which the users are previously registered, happened on the supervised network.

Geo-location Applications (i-Geo): is the application involved in the geo-location of the customers mobile [2]. In particular, our solution locates mobile phones taking into account real and estimated path-loss with all information that can be extracted from a GIS map of the interested area rather than compute the mobile position only through real and estimated path-loss as in classical approaches.

Geographical Electromagnetic Field Information System (GEMFIS): is an open source application used to monitor the network usage focusing on maximizing performance and checking electric pollution levels, in accordance with the current legislation. GEMFIS includes functionalities for storing, displaying and managing environmental data.

In the scenario depicted above, without a SSO solution, the technicians that wished to access Pitagora's tools had to manage several username/password pairs and log-on separately to each service. The adoption of CAS++ solution has brought several advantages. In current Pitagora's architecture, individual services are not stand-alone modules, each with its own access control layer; rather, they are fully integrated in a single security domain. Technicians needing to use multiple applications can perform a single log-on operation and all profile information requested by the application is transparently provided by CAS++. The adoption of CAS++ also improved user profile management, since our profiles repository and administration point are fully integrated within CAS++. Another important requirement fulfilled by CAS++ is strong authentication, a fundamental aspect in our scenario. Finally, CAS++ allowed Siemens developers to freely choose the programming language used to implement individual services.

6 Conclusions

We described some trust models representing different systems behaviors and goals for Single Sign-On services, and identified the requirements that an open source Single Sign-On solution should satisfy. We then illustrated our open source SSO system, called CAS++ and its application to a real case study. Issues to be investigated include an extension of CAS++ to fully support the requirements of a full identity management model.

Acknowledgments

We thank the anonymous reviewers and Tuomas Aura for comments and suggestions which considerably improved the paper. This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

References

1. M. Anisetti, V. Bellandi, E. Damiani, M. Montel, and S. Reale. Open Source Electromagnetic Field Monitoring as e-Government Service. *Proc. of the International Symposium on Telecommunications*, Shiraz, Iran, September 2005.
2. M. Anisetti, V. Bellandi, E. Damiani, and S. Reale. Localize and tracking of mobile antenna in urban environment. *Proc. of the International Symposium on Telecommunications*, Shiraz, Iran, September 2005.
3. C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Towards Privacy-Enhanced Authorization Policies and Languages. *Proc. of the 19th IFIP WG11.3 Working Conference on Data and Application Security*, Nathan Hale Inn, University of Connecticut, Storrs, USA, August 2005.
4. C.A. Ardagna, E. Damiani, F. Frati, and M. Montel. Using Open Source Middleware for Securing e-Gov Applications. *Proc. of the First International Conference on Open Source Systems (OSS 2005)*, Genova, Italy.
5. P. Aubry, V. Mathieu, and J. Marchal. ESUP-Portal: open source Single Sign-On with CAS (Central Authentication Service). *Proc. of EUNIS04 - IT Innovation in a Changing World*, Bled (Slovenia), July 2004
6. C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera. Provisions and obligations in Policy Management and Security Applications. *Proc. of the 28th VLDB Conference*, Honk Kong, China, 2002.
7. D.A. Buell, and R. Sandhu. Identity Management. *IEEE Internet Computing*, November-December 2003.
8. Central Authentication Service, `http://jasigch.princeton.edu:9000/display/CAS`
9. A. Corallo, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, G. Elia, and P. Samarati. Security, Privacy, and Trust in Mobile Systems. *Mobile and Wireless Systems Beyond 3G: Managing New Business Opportunities*, Idea Group Inc., (2005).
10. S. De Capitani di Vimercati, and P. Samarati. *Access control: Policies, models, and mechanisms*, Foundations of Security Analysis and Design, 2001.
11. J. De Clercq. Single sign-on architectures. *International Conference on Infrastructure Security (InfraSec 2002)*, Bristol, UK, October 2002.
12. S. Feldman. The Changing Face of e-Commerce. *IEEE Internet Computing*, 4(3):82–84, May/June (2000).
13. B. Galbraith et al. Professional Web Services Security. Wrox Press, 2002.
14. F. Hao, R. Anderson, and J. Daugman. Combining cryptography with biometrics effectively. Technical report, Cambridge University - Computer Laboratory Technical Report UCAM-CL-TR-640.
15. Java Open Single Sign-On (JOSSO), `http://www.josso.org/`.
16. R. Khosla, E. Damiani, and W. Grosky. Human-Centered E-Business. Kluwer Academic Publishers, Massachusetts, USA, 315 pages, April 2003.
17. Liberty Alliance Project, `http://www.projectliberty.org/`

18. PRIME (Privacy and Identity Management for Europe), http://www.prime-project.eu.org.
19. Shibboleth Project, http://shibboleth.internet2.edu/.
20. Single Sign-On, The Open Group, http://www.opengroup.org/security/sso/.
21. SourceID Open Source Federated Identity Management, http://www.sourceid.org/index.html