A Trust Negotiation Based Security Framework for Service Provisioning in Load-Balancing Clusters

Antonio Maña, Hristo Koshutanski*, Ernesto J. Pérez

Computer Science Department, University of Malaga,
E.T.S. Ingenieria Informatica, Campus de Teatinos,
Malaga 29071, Spain

Abstract

The OKKAM project aims at enabling the Web of Entities, a global digital space for publishing and managing information about entities. The project provides a scalable and sustainable infrastructure, called the Entity Name System (ENS), for the systematic reuse of global and unique entity identifiers. The ENS provides a collection of core services supporting entity identifiers pervasive reuse. The ENS is required to be reliable data intensive load-balancing cluster system for service provisioning.
Given the project's successful outcome, this paper presents the ENS security framework and how it enables scalable secure service provisioning underpinned by trust negotiation based access control. A detailed security performance evaluation is given, with supporting conclusions of scalable and efficient security design and implementation.

Keywords: Access control, trust negotiation, web services security, security proxy, security evaluation, load-balancing cluster.

1. Introduction

The OKKAM project[1] aims at enabling the Web of Entities, a global digital space for publishing and managing information about entities, where each entity is uniquely identified, and links between entities can be explicitly specified and exploited in a variety of scenarios. Compared to the WWW, the main differences are that the domain of entities is extended beyond the realm of digital resources to include objects in other realms such as products, organizations, associations, countries, events, publications, hotels or people; and that links between entities are extended beyond hyperlinks to include virtually any type of relationship [1].

The concept of the "Internet of Things"[2] (IoT), widely referred to as a world-wide network of interconnected objects, shares several similarities with the concept of the Web of Entities. Compared to the IoT, the Web of Entities is a step towards the realization of a IoT where the domain of entities and links between them are extended to allow any type of virtual, physical or digital relations.

To make this happen, the Web of Entities must trigger (as the WWW did) what economists call the network externality effect, and this requires at least three pillars to be in place:

*Corresponding author. Postal address: E.T.S. Ingenieria Informatica, Campus de Teatinos, Malaga 29071, Spain. Telephone: +34-952-133303, Fax: +34-952-131397

Email addresses: amg@lcc.uma.es (Antonio Maña), hristo@lcc.uma.es (Hristo Koshutanski), ernestopg@lcc.uma.es (Ernesto J. Pérez)

[1] http://www.okkam.org; http://community.okkam.org

[2] http://www.itu.int/internetofthings/

- a suitable infrastructure which can support the open and sustainable growth of the Web of Entities – calling for a *security architecture* design enabling future infrastructure evolution;
- a critical mass of new entity-aware content and data accessible to a very large number of users in a relatively short time period – calling for *efficient security access* to core services;
- a collection of exemplary and high impact applications, which can prove to the key players in ICT that investing in the Web of Entities is worthwhile – calling for appropriate *security engineering* for users and application developers.

The OKKAM project presents a strategy and an effective work plan to build the three pillars of this Web of Entities, not only from a technical point of view, but also from a social, organizational and business-oriented perspective. The project provides a scalable and sustainable infrastructure, called the Entity Name System (ENS), for systematic reuse of global and unique entity identifiers [2, 1, 3]. The ENS stores identifiers for entities and provides a collection of core services needed to support their pervasive reuse.

From a technical perspective, the ENS is required to be a highly reliable data intensive load-balancing cluster system for service provisioning. As such, the ENS is an excellent target to demonstrate a solution for providing security and trust aspects of service usage. Taking that into account, the paper reports the experience, going from the design of the security architecture down to its implementation, and describes security solutions developed as the final outcome of the project.

The ENS offers a collection of core services accessible via Web Services technologies. Due to the nature of the ENS content (identifiers of entities) and its service to the general public with decentralized service usage, there is a strong legal requirement for controlling who creates, modifies, or administers entities' profiles in the ENS. On the other hand, there is an important user privacy requirement of not only controlling who uses the ENS but also ensuring privacy of users' contributions, i.e., no third party (other than a user and the ENS) should learn by itself who creates, modifies, or administers the ENS, i.e. confidentiality and authenticity of ENS service access.

Nowadays, there are a number of standards addressing generic security and trust aspects for Web Services (known as WS-*), which calls for a specific security design deploying those standards into a comprehensive security framework, especially tailored to OKKAM, with easy to use security mechanisms for practical adoption.

Trust, security and privacy approach, undertaken in OKKAM, is mainly determined by the open and decentralized interactions by a community of users. The public nature of OKKAM is that it serves the public and, at the same time, is open to contribution by the public. The decentralized style of OKKAM is determined by the fact that the ENS stays "hidden" to end users' interactions. End users interact with the ENS via OKKAM Web front-ends.

Secure and trusted usage of the core ENS services is driven by access control based on digital certificates attesting to users' privileges (credentials). Credentials themselves convey sensitive information and may become subject to unauthorized misused and disclosure. Recent years have seen the emergence of a concept called trust negotiation [4, 5]. It is a policy-based technique that provides clients with the right to protect their own credentials and to negotiate with servers access to those credentials. Trust negotiation preserves peer's privacy by controlling what credentials are disclosed to what entities under given conditions. Thus, trust negotiation allows users and the ENS to mutually establish confidence in each other by requesting credentials until sufficient trust is established to grant service access.

Access control is driven by a trust negotiation mechanism triggered automatically whenever a client needs more access rights to use a service. The goal is to provide flexible and efficient access control enforcement that fits in well with the OKKAM aim of providing services to an open/public community. The negotiation approach also facilitates future evolution of ENSs in

terms of security requirements organizations and companies may have over time, and uniform automated enforcement of those with user-side applications.

The main achievements of the security framework are: (i) Design of a trust negotiation mechanism meeting the requirements of the ENS in terms of performance and scalability. The trust negotiation mechanism improves on the work of [6, 7] achieving *server-side stateless negotiation* for true replication and distributed session management of policy decision point (PDP) component within a cluster of machines. (ii) Development of *security proxy* leveraging secure and trusted communication between user applications and the ENS. The security proxy encapsulates automated trust negotiation functionalities with the Web services security technologies providing high-level security abstraction. (iii) A detailed security performance evaluation supporting conclusions of scalable and efficient security process design and implementation.

The rest of the paper is organized as following. Section 2 presents related work on access control systems. Section 3 overviews the ENS architecture and main components. Section 4 presents the security requirements defined at the start of the project. Section 5 presents the design decisions underpinning the security architecture and what motivates them. Section 6 presents the underlying negotiation model of access control. Section 7 discusses implementation details of the security architecture, interactions, and deployment. Section 8 reports the results of performed security evaluation. Section 9 discusses security management aspects of the system and its robustness analysis. Section 10 concludes the paper and outlines future work. Appendix A presents details of the designed OKKAM certification and attribute management model.

2. Related Work

We believe that an important aspect of an authorization system for any open environment is the incorporation of a trust negotiation capability. The focus of this paper is on the development of an authorization system with integrated trust negotiation capability by emphasizing on its architectural design and negotiation enforcement aspects meeting the security requirements. One of the main architectural features of distributed access control systems is splitting the server role into two: an *application server* and an *authorization server*, to decouple access control logic from application logic and possibly distribute/replicate the access control component.

One of the earliest papers on providing a general framework for expressing authorizations was proposed by Woo and Lam [8]. The main component of the system is an authorization server that performs authorization on behalf of an application server. An application server elects an authorization server in order to offload its access control policy for further evaluation. The authorization server takes the final access decision and hands out authorization certificates to authorized clients. These certificates are to be forwarded by clients to the application server together with their requests.

Akenti [9] is an authorization systems based on digital certificates: X.509 user identity certificates for authenticating users; use-condition certificates for specifying the conditions that must be met by a user to gain access to a resource; attribute certificates, stored on trusted servers, attesting that a user possesses specific attributes.

Adage system [10] offers centralized security administration and modular authorization. An application server communicates with an authorization server for obtaining authorization decisions. The authorization server in turn communicates with identity and attribute servers to get additional information for a client. In the process of decision making the authorization server determines whether the user needs some roles to be activated and attempts to activate them.

PERMIS [11] adopts X.509 certificate standard of identity and attribute certificates. Architecturally, it consists of two main subsystems: the privilege allocation and the privilege verification subsystem. The former is responsible for assigning privileges to users – issuing X.509 role assignment attribute certificates, as well as, signing and issuing a service provider's access policy as an X.509 attribute certificate. The privilege allocation subsystem stores its attribute certificates in LDAP directory for subsequent use by the privilege verification subsystem. The

privilege verification subsystem authenticates and authorizes a remote client as well as takes access decisions on target services.

The access decision process of Akenti, Adage and PERMIS systems is based on identity, authorization, and attribute data stored and conveyed in certificates dispersed over the Internet (e.g., LDAP directories, Web servers). The authorization engine has to gather and verify the certificates needed for the user's request and then evaluate them to compute an access decision.

In our approach we also split the ENS application logic from the underlying security logic by means of the security proxy. We replicate the security proxy throughout the replication of ENS nodes in a cluster to achieve scalability of security services. However, by design, our system does not look on its own for available client's certificates when taking a decision but allows users control what credentials they are willing to use for service access via an automated negotiation process.

2.1. Trust Negotiation Based Access Control Systems

Trust negotiation [12, 13] is a promising technique that has been proposed for authorization and access control in open environments where any entity could be malicious. Trust negotiation allows authorization of peers which have incomplete knowledge about each other, belong to different administrative domains, and may never have interacted before. Trust negotiation implies authorization based on bilateral exchange of credentials. The sequence of credential exchange during a negotiation process is controlled by negotiation strategies. One can define a variety of negotiation strategies and privacy settings [14, 15, 16, 17] depending on credential sensitivity, on familiarity with the remote opponent or domain, type of resources being accessed, and so on.

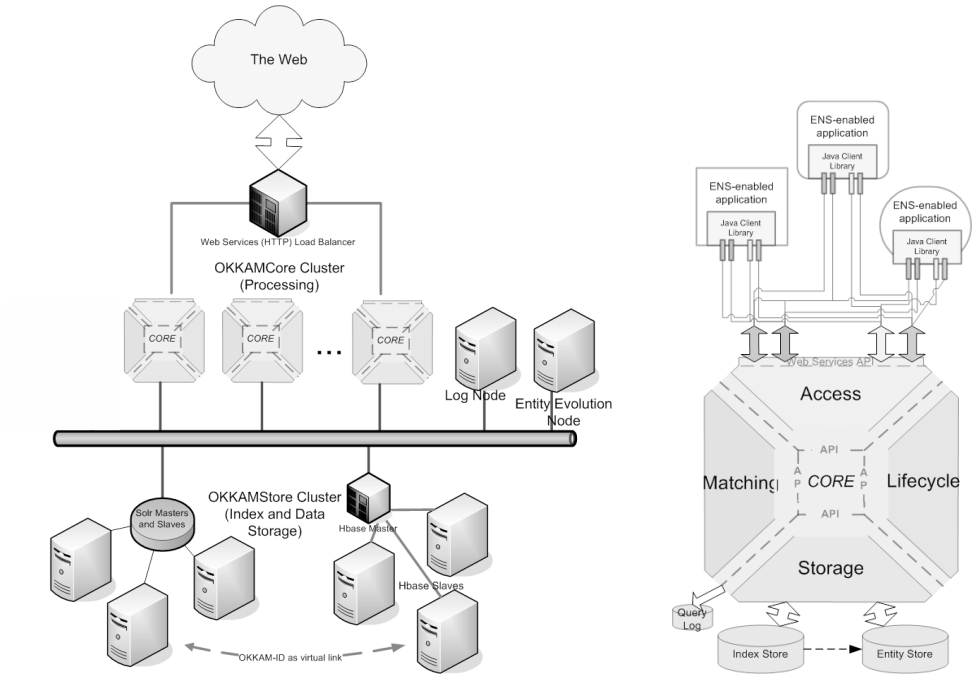
Recently, only a few policy-driven trust negotiation approaches have been successfully developed as authorization systems, such as TrustBuilder2 [18], Protune [19], and Trust-X [20]. Traust authorization service [21] incorporates into GridFTP service a trust negotiation functionality using existing negotiation mechanisms and protocols such as those of TrustBuilder and Trust-X.

An approach [22] proposes a fine-grained Grid monitoring system integrated with the iAccess negotiation system to enable comprehensive protection of Grid computational resources from the threat represented by applications executed on behalf of remote Grid users.

Instead of using centralized trust negotiation service, the model proposed in [23] deploys PeerTrust [24] negotiation capabilities into the Grid service functionality. A policy decision point (PDP) of Globus authorization service is extended by an intercepter of service requests. The intercepter grants access to a service if a negotiation process has been successfully completed. Negotiations are asynchronous and implemented through WS-Notification mechanisms. The implementation changes the service WSDL file and exploits state-full resources defined by WS-Resource framework.

The focus of this work is on the development of an authorization system with improved trust negotiation capability of [6, 7]. We refer the reader to [6] for a detailed comparison of the underlying negotiation approach with the state-of-the art. Our solution implements synchronous client-initiated negotiations where the server agent is free from negotiation state maintenance. Our argumentation is that stateless server-side negotiation enhances system robustness against potential denial of service attacks with respect to the stateful negotiation approaches (see also Section 9).

The ENS security architecture is designed to handle multiple trust negotiations for decision making by using distributed access control session data. The new negotiation design allows for replication of the PDP service on several machines within a cluster for a scalable decision making process. In this way, if a machine where a negotiation process started is not capable of handling more computations, the load balancer will redirect the coming steps of the process to another node in the cluster where the negotiation process can continue smoothly without repeating the steps previously taken. This is an important advancement towards a flexible access control process for a distributed (grid) computing scenario, where an access control process dynamically



(a) ENS Multi-core Cluster Architecture

(b) ENS Single-core Interactions View

Figure 1: ENS V2 Service Production Platform [25]

on the fly spans/continues from one computing node to another one, without intercepting the access control process.

One could also extend the security proxy component to fetch relevant certificates from dedicated repositories on the Web which are actually necessary for a current negotiation step. In that way, users run the security proxy locally and configure it to obtain user certificates from remote LDAP repositories where users (prefer to) keep their credentials.

3. ENS Architecture Overview

We will first overview the ENS architecture and main components in order to better introduce security requirements and the ENS design decisions. The ENS system has been defined to provide core services with a high degree of reliability. For that reason, a dedicated high-reliability cluster architecture has been designed with its respective database technology addressing reliable storage and provision of ENS data [25, 26, 2].

Figure 1 shows the ENS as (a) a multi-core cluster view and (b) interaction of ENS with external application as a single-core view. The ENS Core cluster aims at providing high-level processing of the core ENS services by replicating ENS core functionalities on multiple machines (in a local area network) all sitting behind a load balancer.

The ENS core components illustrated in Figure 1(b) are briefly described below:

- Storage component: stores IDs with the corresponding profiles and provides rich indexes for efficient retrieval.
- Matching component: uses a library of methods to match any arbitrary description of an entity to its ID (if any) in the ENS repository.
- Lifecycle component: controls the evolution of the ENS repository through time and space.

- Access Manager: responsible for proper access control of function calls and dispatching of internal messages from one layer of the core to another.
- OKKAM Web Services: provides general APIs allowing authorized users of the application to access and interact with the ENS.
- ENS-enabled applications: any application which can use the interfaces to interact with the ENS.

An application perspective means that an application communicates with the cluster as if it were communicating with a single-server machine (Figure 1(b)). An important factor (and architectural decision) here is that the load balancer does session-less dispatching to any of the cores behind. This decision ensures availability and scalability of the ENS services and defines an important requirement for the design of the security architecture.

A single core behind the load balancer has a set of Web Services APIs used by external applications to access the ENS. There is an explicit layer in the single-core view architecture, called Access Manager, responsible for properly dispatching function calls (behind the APIs) and internal messages from one layer of the core to another. We refer the reader to [25, 26] for further details on the ENS architecture and storage layer design features, and to [2, 27] for recent results on the ENS functional aspects and scalability performance. The important aspect here is that security and trust solutions have been placed at the level of the Access Manager as the main layer where all communications both external and internal between different layers are processed.

4. Security Requirements

There are several security requirements defined at the start of the project that a security solution should conform to. We classify them into security architecture design, software engineering, and performance-driven requirements.

There are two main design requirements behind the security architecture:

- Define a "backbone" of secure and trusted communications with ENS so that any evolution of ENS and future decision making are seamlessly supported without any major change of the security architecture. This necessity is underlined by a general OKKAM design goal of making the ENS flexible and evolvable over time.
- Adopt well-known and widely used security and trust standards as building components of the security infrastructure, and employ maximum possible portion of each standard. Thus, maintaining important balance between targeting the best possible solutions beyond state-of-the-art, which require a bigger deployment effort, and achieving interoperable standards-based communications with the ENS.

It has been defined the need of core security requirements of confidentiality, integrity, authentication and access control that external components should implement for all communications with the ENS. From a software engineering point of view, a set of security solutions covering the above requirements should be properly integrated into OKKAM-enabled applications during the application development process.

However, it is important to note that just providing security solutions to OKKAM-enabled applications is not satisfactory as it still leaves in the hands of the application developer the burden of understanding what a security requirement means in a given context and if that security requirement is exactly what a developer needs for a given application or a combination of them. We believe that an application developer may have difficulties in understanding how to integrate core security solutions (with each other) to achieve correct security for his applications, unless the developer has a good background in security.

For that reason, it has been decided to analyze the necessities of OKKAM applications further and provide the most needed security aspects as an integrated security solution (of core security requirements) to use as a high-level security property for application developers, ENS developers, and end users. In what follows we list some of the requirements of OKKAM related to software engineering of OKKAM:

- Developers of OKKAM-enabled applications should not need to have specific knowledge of underlying security properties in order to enable secure communications between their applications and the ENS,
- Developers of the ENS system (its functional aspects) should not be required to know the internal components of available security solutions in order to protect ENS services and to apply those security solutions inside the service development of the ENS,
- The design of a security solution that provides the highest possible abstraction of security aspects to facilitate easy and smooth integration on the application level, thus addressing ENSs future evolution.

There is a legal requirement to trace who creates and manages ENS entities. This requirement has two aspects that need to be addressed: trace who creates and manages ENS entities by direct interactions with the ENS services; and trace those end users who create and manage ENS entities via indirect (third-party) service interactions with the core ENS services. This last aspect is defined by the general project goal of allowing third party community services to offer meta services to end users based on core ENS functionalities.

The ENS requires an efficient security process for ENS core services in order to cope with expected impact and wide adoption. There are two main security-related performance requirements pursued regarding usability:

- A reasonable (user's perception) delay caused by mutual authentication (initial handshake) between a user application and the ENS should be no more than a second.
- A reasonable security-related overhead once the mutual authentication is completed should not exceed a few hundreds of a second per service access.

The first performance requirement regards the aspect of providing accessible for ENS nature bilateral authentication process leveraging subsequent Web service communications. The second requirement concerns two ENS use cases: preserving as much as possibly ENS capacity for handling search queries per second over a security channel, and providing as efficiently as possible bulk import access over a security channel. The bulk import setting reflects the efficiency of automated tools used for importing large data sets of entities into the ENS repository.

5. Security Architecture Design Decisions

There are several design decisions taken about the security architecture based on the requirements explained above. This section shows the motivation and justification of the decisions taken.

Figure 2 shows a high-level view of ENS community interactions. There are several actors that interact with the ENS. Web services applications represent all third-party information systems that integrate ENS services into their business logic. In this category we also consider bulk import tools developed by the project consortium, as well as, private ENS nodes which serve OKKAM services entirely in a private company-specific domain of entities, for example, in private company products. There are ENS-empowered tools, developed by the project consortium, dedicated to help end users using the ENS from within other applications (such as MS Word plug-in or Firefox

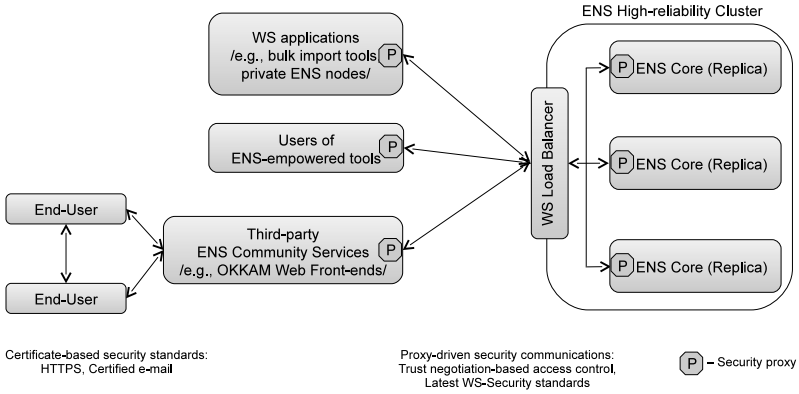


Figure 2: OKKAM Security Architecture and Communications

plug-in etc.). Some of the tools in this category do not have dedicated capacity of processing any security settings.

The third category of actors represents third party ENS community services which provide ENS-based services to end users. Some dedicated community services are provided by the OKKAM consortium such as the Web front-end toolkit which facilitates end users with easy and intuitive management of ENS content.

Another important aspect of the security architecture is the fact that the ENS is designed to work as a high-availability cluster with several ENS node replicas behind a load balancer component. This aspect of node replication has an important influence on the choice of security standards and on the design of the access control decision and enforcement components.

WS-Security standard[3] defines end-to-end interoperable message content integrity and confidentiality when messages pass through intermediary nodes before reaching their destination end-point. This aspect fits in well with the choice of having a load balancer allow application-to-ENS-node-replica secure channel establishment as shown in Figure 2. In such cases, the load balancer is free of handling secure channel cryptographic operations, as it would be in the case of TLS connections, instead, each ENS node handles the security operations.

5.1. Certificate-based Security

Security and trust aspects of the OKKAM infrastructure focus on controlling access to information stored on the ENS. The security architecture is based on the advanced use of certificate technologies with the strategic goal of fostering the creation of a wide community of users by providing them with the full benefits of certificate technologies such as the capability of establishing trusted and secure communications with ENS services, confidential communications among end users, and third party ENS service providers, as well as, document authentication based on digital signatures.

Trust in the ENS community is based on certificate authorities that qualify the ENS, third party service providers and end users by means of digital identity and attribute certificates compliant with X.509 [28] standard. The architecture allows scalable trust establishment in a distributed environment by means of common trusted OKKAM certificate authorities. Appendix A presents details of the designed OKKAM Public Key Infrastructure, certificate authorities and attribute management.

[3]Web Services Security Specifications at `http://www.oasis-open.org/specs`

5.2. Proxy-driven Security Communications

Communications with ENS cluster are based on Web Services technologies. However, securing Web services interactions and building a scalable authentication and access control process is a non-trivial task, which involves the usage of several web services security standards and proper synergy of them. In addition, certificate management and scalable negotiation-based access control add another layer of complexity in communications with the ENS.

To aid users, developers and third-party service providers a *security proxy* component has been designed that abstracts all necessary security management and technological aspects from application-level development. Figure 2 shows how the proxy component enables client-side secure and trusted communications with the ENS, and how the architecture realizes scalable security processes by deploying the proxy counter part in any ENS core in a cluster.

The proxy component enables user-side uniform management of security, trust and privacy settings of communications with both the ENS and private ENS nodes. It also enables private (proprietary) ENS nodes to seamlessly establish secure and trusted communications with the public ENS. The latter is defined in the context of a query forwarding functionality where a private ENS node may connect to the public ENS for complementary information on given entities from the public space.

5.3. ENS Security Mechanism

The main steps taken by the security mechanism upon a service request to the ENS are as follows:

1. Establishment of a confidential channel between client proxy and the ENS (via asymmetric crypto operations).
2. Trust establishment on more access rights (when necessary) to access a requested service. Trust negotiation over the confidential channel (symmetric message encryption with asymmetric crypto operations on certificate verification).
3. Any subsequent access to ENS APIs is over the confidential channel of step 1 (symmetric message encryption).

An important aspect of the ENS security design is the dynamic access control enforcement when accessing ENS services. This means that within a given security session if a user needs more access rights for a given service, a trust negotiation process is triggered on the fly between the user-side proxy and the ENS (transparently for the application-level), and upon successful agreement the given operation service is automatically invoked. Steps 1 and 2 of ENS security take place automatically by a means of a priori installed certificates. The ENS plays the role of Policy Decision Point (PDP) for authorization.

The ENS security solution takes advantage of ENS being a high-reliability cluster and builds all security processing as scalable as the ENS itself by replicating security services throughout the replication of ENS core services.

5.4. ENS Security Mechanism vs Single Sign-on Mechanism

We will motivate the design decisions on the ENS security mechanism derived from analysis of the widely used single sign-on (SSO) mechanism [29][4]. SSO is a system design approach to share user authentication between different administrative domains. There is a central identity provider (IdP) domain that authenticates users so that they can smoothly access all the services of the providers (SPs) sharing trust with that IdP. There is a circle of trust formed between service providers, which require all of them to have similar levels of trust. A user presents its username/password only once per session, and for the duration of the session the user is

[4]The main SSO steps are taken from the SAML specification of Web browser single sign-on schema [29].

Features	Single Sign-on	ENS Security
Performance of user authentication	Degrades with increasing number of registered users due to underlying database lookups for verification.	Independent to number of users and location as the verification is done at the level of certificates (without database lookup).
Scalability of multi-user authentication	Requires additional infrastructure investment to enable scalable authentication for multi-user access.	Inherits scalability of ENS multi-cluster architecture (security services replicated on any ENS core).
Availability of authentication service	Requires additional infrastructure investment to enable authentication service availability.	Inherits availability of ENS services (security services run on any ENS core).
Usability of authentication service	Good usability level when different administrative domains federate for service access.	Good usability level for OKKAM purposes: decentralized authentication and minimized username/password management.
Users trust in security technologies adopted	SSO constitutes a good choice for distributed systems requiring uniform authentication scheme and strong trust relationships between federated sites.	Certificates provide good security level of users' data; minimize username/password management; provide good perception for decentralized authentication and authorization. Proxy-based security gives good balance of simplicity of use and security level achieved; gives perception of well-thought out security design for better integration (adoption) of ENS in other information systems.
Independency of external security resources	Availability, performance and manageability depend on external IdP-specific authentication resources.	Availability, performance and manageability depend on ENS own and well-controlled security resources (own CAs, user authentication, proper certificate generation, security policies, etc).
Cryptographic operations used	– Asymmetric for establishing confidential channel (e.g., TLS) for users presenting credentials to IdP. – Asymmetric for establishing confidential channel (e.g., TLS) for user presenting IdP's authentication token to SP, and for SP validating the token. – Symmetric encryption for subsequent interactions between users and SPs.	– Asymmetric for establishing confidential channel (mutual certificate security). – Asymmetric for trust establishment phase (certificate verification and validation) run over confidential channel. – Symmetric encryption for subsequent interactions with the ENS (WS-Security for SOAP message encryption).
Certificate revocation	SSO proper management is dependent on certificate revocation. For example, HTTPS channels require a user-side certificate revocation check on servers' certificates.	Certificate revocation status required on both sides client- and server-side when establishing confidential channel, and during the trust negotiation process.

Table 1: Security features and comparison with SSO and ENS security

authenticated for all SPs by presenting the obtained security token from the IdP (without re-entering its credentials).

The main steps of the SSO mechanism upon receiving the very first service request by a user:

1. A user is forwarded (by a SP) to the respective IdP of the circle of trust.
2. Establishment of confidential channel between the user and the IdP domain (via asymmetric crypto operations, such as HTTPS).
3. The user presents their username/password credentials for authenticating to the IdP. The

IdP generates an authentication token (e.g., SAML assertion) and digitally signs it.
4. The IdP passes the user back to the SP's domain with the authentication assertion. There is an intermediary step where the user establishes a secure channel with the SP for supplying the authentication token (normally HTTPS), otherwise the user should prove to the SP he is the owner of the token (involving some form of cryptographic operations).
5. Any subsequent access to the SP's domain is performed either on a public or on a confidential channel (symmetric message encryption). In the context of ENS this is necessary to ensure confidentially and privacy of users' contributions.

Table 1 summarizes the most relevant security features of the project and how they contrast with SSO and the ENS security mechanism. Regarding SSO performance, we refer to the results of DAMe project. One of the project goals is to realize a unified SSO framework on top of the eduroam[5] infrastructure. In a project deliverable [30, page 46] the authors report that for their SSO application scenario the IdP takes on average 0.5 of a second to validate if a user is among a legal set of users and to generate a digitally signed authentication assertion (a SAML assertion). There are also other experimental results such as, an authentication time can reach 22 seconds in the simplest case and 32 seconds in the extended authentication mode. The evaluation results show that SSO performance depends heavily on the number of registered users and the scalability of IdP's SSO services.

6. Trust Negotiation Based Access Control

The trust negotiation mechanism developed in the OKKAM security framework improves on the work done in [6, 7] in order to cope with the needs of the ENS. The approach in [6] proposes an interactive access control model (IAC) where a server interacts with a client requesting from him missing credentials necessary to grant the service. The client, in turn, checks if it has the requested credentials and sends a response back to the server. The server re-evaluates an access policy to verify if the returned set of credentials grants access to the object. In case the client does not have all the credentials from the first round, the server re-computes a new set of missing credentials and asks for them from the client. Thus, a client and a server interact until either the client presents a set of credentials satisfying the server's access policy or there is no missing set to be asked for from the client and the server denies access.

The work in [7] defines an extension to IAC where a client entity is also empowered by having its own IAC reasoning so that whenever a server asks a client for a set of missing credentials the client computes, according to its own credential control policy, what missing credentials the server has to present to see the client's credentials. The extension defines a negotiation protocol, on top of the IAC model, allowing a client and a server to interact until an agreement is reached and the server provides access to the requested object, or one of the parties denies the negotiation process; failing to provide the required credentials.

6.1. Trust Negotiation Design Decisions

The negotiation protocol has been redesigned and substantially improved to meet the ENS system requirements allowing decentralized computing of negotiation steps within a cluster of machines. Looking at the ENS architecture, there are three main aspects to be addressed: (i) integration of the trust negotiation protocol with Web Services security standards, (ii) enable "passive" server-side trust negotiation process due to client-side firewall restrictions and inability to have direct server-to-client requests, (iii) allow session-free and cryptographic-free load balancer of the ENS cluster thus achieving truly functional load balancing and distributed computing within the cluster (of machines).

[5] `http://www.eduroam.org`

```
InteractiveAccessControlStepwise(r,Cp){
1. check if PA and Cp entail r,
2. if the check succeeds then return access granted
3. else
    (a) compute a set of disclosable credentials CD entailed by PD and Cp ,
    (b) compute a set of missing credentials CM subset of the disclosable ones such that:
       - PA together with Cp and CM entail r, and
       - PA together with Cp and CM preserve consistency
    (c) if CM does not exist then return access denied
    (d) if CM exists
        i.   compute a stepwise missing set of credentials
             CM1 = doStepwiseDisclosure(CM)
        ii.  if CM1 exists then return CM1
        iii. else return access denied
}
```

Figure 3: Interactive Access Control Algorithm with Stepwise Disclosure

The negotiation protocol in [7] focused on providing efficient negotiation process in agent-based autonomic communications. As such, the protocol forces any two opponents to be stateful by keeping active state of a current negotiation process in order to consistently request missing credentials to each other. There are two main disadvantages of the negotiation protocol:

- The protocol design requires a direct and open network connection between negotiation agents enabling direct requests between a client and a server. This aspect limits protocol usage especially in Web services-based systems where, for example, the presence of firewall proxies disables direct connections to clients.
- There is no track of requests and counter-requests on the opponent side. An opponent cannot distinguish when a credential request comes in response to a counter-request and, as a consequence, a negotiation deadlock[6] situation is indistinguishable from a long negotiation process. In this case, if another negotiation path is possible then in the presence of a deadlock the session will expire and the negotiation process will fail, even if a successful negotiation alternative exists .

The new design goal is to achieve a *stateless* negotiation process on the server side, while keeping *stateful* negotiation on the client side. Thus, a client agent can establish negotiations with a server agent only via request-response messages traversing a firewall proxy and load balancer.

We improved the negotiation protocol of [7] by making the core interactive access control model of [6] *step-wise*, i.e. instead of returning a set of missing credentials that satisfy a given request, the IAC algorithm returns a stepwise set of credentials, subset of the missing set, by strictly following the disclosure policy information. The enhanced interactive algorithm is used on *both* client and server sides for decision making.

Figure 3 shows the interactive access control process enhanced with the stepwise disclosure control functionality. Here, `PA` denotes access control policy of an agent, `PD` denotes credential disclosure policy of an agent, `Cp` denotes all credentials presented by a remote opponent within a given access control session, and `r` denotes the service request a negotiation is triggered upon.

Steps `1`–`3d` are those of the interactive access control [6, page 12] while steps `3di`–`3diii` have been taken from the original negotiation protocol [7, page 16]. However, there is an important element to take into account when designing the new access control algorithm with respect to the old IAC model of [6] – we have to internally transform `PD` to be credential step-wise aware policy with respect to the client's *declined* (to present to the server) credentials. Following that,

[6]Negotiation deadlock is for example when a server requires credential A for a resource, but disclosing credential A at the client requires first credential B disclosure from a server, but credential B on the server requires credential A on the client. In negotiation deadlock neither of the opponents progresses and the negotiation fails.

step 3a computes disclosable credentials if they are entailed by a valid credential chain having no declined credentials in its path. In that case, we achieved *consistent* computation of a step-wise set of missing credentials from step 3di based on the computed set of missing credentials from step 3b for all client-to-server stateless negotiation interactions within a security session.

We make PD step-wise aware (step 3a) following the method described in [7, page 16]. Informally, we syntactically change the structure of PD by making any credential term in a head of a rule deducible by PD if it has not been declined by the client[7]. Important to note is that the modified stepwise-aware PD remains a stratified logic program if the original PD is stratified, which is a prerequisite for the IAC model [6].

Server-side guarantees. Trust negotiation algorithm on the server side has been replaced by the new interactive access control algorithm with a stepwise disclosure control. We removed negotiation state maintenance from the server side, thus creating a stateless negotiation service.

The new server-side negotiation process guarantees that if a negotiation starts with a given set of missing credentials (necessary for granting a resource), over subsequent interactions the same set of missing credentials will be recomputed by the server for the given request and the stepwise approach will progress with the correct negotiation state. We recall that to guarantee consistent stateless negotiation process PA has to be a well-behaved policy and PD a stratified logic program. We refer to [6] for details of both policy requirements and point out that these policy requirements do not limit policy expressivity but do cover a wide range of possible access control specifications including those of the ENS.

Although the server-side algorithm guarantees a consistent negotiation process over stateless Web services communications, yet the protocol has to ensure that a client does not gain more information than allowed by a server whatever negotiation state manipulations a client may perform. To address this, we keep the following information on the access control session data on the server side: client's presented credentials, client's declined credentials, and *history* of credentials requested to the client within a session. That information helps us to detect:

- if a client wants to make the server's negotiation process compute past negotiation results (repeating past steps) thus forcing endless negotiation processes (not gaining more information but) making easy a denial of service attack,
- when a client wants to inject non server-requested credentials thus maliciously probing server's behavior.

On the one hand, the server-side modification allows us to simplify the negotiation model by removing active negotiation state maintenance and, on the other hand, to preserve flexible disclosure control of credential information to an opponent and the same negotiation protection of credential disclosure as in the old negotiation protocol [7].

We added a new feature on the server-side negotiation that if a client presents non-server-requested credentials (for whatever reason) the server agent will deny the negotiation process for a service access. This feature gives some protection from malicious probing by clients wishing to explore the server's behavior and potential DoS attacks.

Client-side guarantees. Client-side negotiation schema also adopted the IAC with stepwise disclosure control, and transformed the old negotiation algorithm from a thread-based implementation to a recursive execution of negotiation steps. This last choice was influenced by the fact that we wanted to detect situations of negotiation deadlock, a disadvantage of the old protocol design, allowing whenever possible successful negotiation between two opponents. We achieved comprehensive negotiation via a recursive process where the client agent keeps stateful nested negotiations and unrolling them back consistently. In that way, the client-side agent keeps track

[7]The access control session data [6] keeps a set of declined credentials by a client not shown in the figure for simplicity.

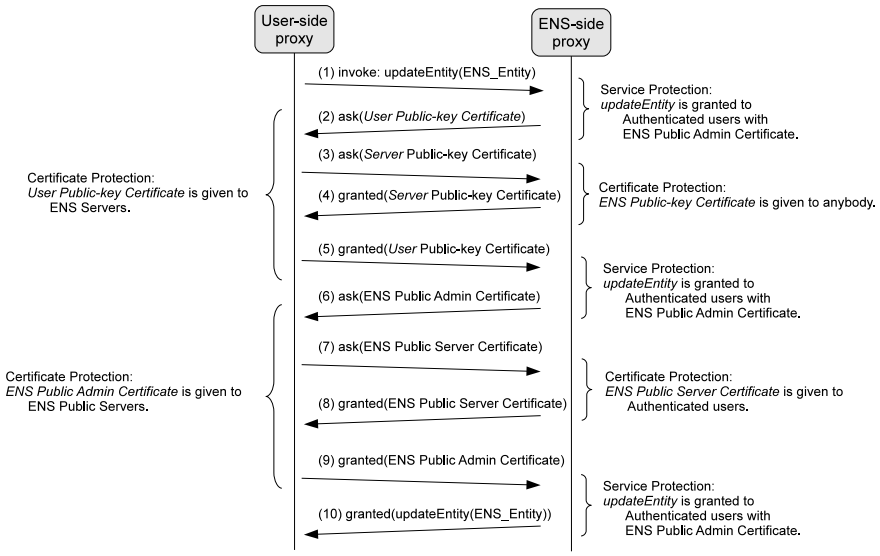


Figure 4: Trust Negotiation Example

of current negotiation status and in the presence of deadlock denies current negotiation step (credential request) to enable further progress and potential negotiation completion.

The new client-side negotiation schema also allows us to traverse firewalled networks and a load balancer via client-initiated request-response communications. This was an important step towards Web Services communications.

6.2. Trust Negotiation Example

Trust negotiation puts in the hands of the clients important protection of their own credentials (certificates) when interacting with ENS, where a user may have credentials for both public ENS and private (company-specific) ENS. In such a case, a user can define fine-grained disclosure control of own credentials and depending on the remote ENS a user interacts with, the proxy will disclose only relevant credentials.

Figure 4 shows a scenario of the administrator privilege establishment necessary for granting exclusive access to update entity service of the ENS. On the left-hand side and on the right-hand side of the negotiation process the policy rule governing a current negotiation step is shown. For example, the policy rule for update entity service states that access to that service is given to legal OKKAM users who have an administrator privilege.

When a user sends an update entity service request, the ENS server-side proxy requests the user to authenticate as a registered user. On the user side, the proxy component looks for a rule protecting the user public-key certificate necessary for user authentication. The user-side security policy specifies that the user's public-key certificate is given only to entities identified as ENS servers. In turn, the client proxy sends a counter requests for server authentication as ENS server. The server looks at its security policy and grants its public-key certificate to the client. Next, a mutual authentication process follows with session key establishment based on mutual certificate security standard.

The steps of mutual authentication have been designed to be taken for all protected ENS services (with proof of possession of public-key certificates). We achieve that by using WS-SecurityPolicy standard, in the WSDL document of the protected WS APIs, specifying all services as protected access with user-side authentication by means of X.509 token format. In addition, the latest Metro release enables, by using WS-SecurityPolicy, servers to publish inside the WSDL of protected services, their public-key certificates so that any client parsing the

WSDL can obtain all the necessary information for starting up the mutual certificate security protocol. For the time being, the client-side proxy first obtains the server identity certificate via a dedicated WS API and then performs the mutual certificate security mechanism.

Once client and server complete a successful authentication, the information on presented certificates is made available on the access control level to facilitate the decision making process. Next, the access control module at the ENS side computes that the user needs an administrator attribute certificate to access update entity service, and requests that from the client.

In turn, the client proxy consults its security policy to check whether the client's administrator certificate is protected. The policy says that the user administrator certificate is authorized for disclosure only to ENS servers run as ENS Public. This requirement is defined since the user has administrator rights only on the ENS (not on ENS private nodes). In that case, the client proxy negotiates the disclosure of the user's administrator right with the server proxy. Since the user is already authenticated, the access control mechanism on the server side grants the disclosure of server's ENS Public attribute certificate to the client. Next, the client discloses its administrator certificate to the server and the server grants an invoke operation on the service update entity.

6.3. Brave and Cautious Negotiation Modes

Another novelty of the negotiation protocol regarding performance and practical aspects is that it enables two explicit modes of negotiation: brave and cautious modes. Any client-server negotiation process takes place over a secure communication channel and with the verified identity of the opponent.

Cautious negotiation mode enables access control decision process on any credential request by an opponent. In other words, this mode enables policy-driven negotiation. Referring to the example above, when the ENS requests an administrator credential from a client, the client proxy triggers a decision making process if the credential policy defines any protection on disclosing the requested credential, and then grants or negotiates with the server on missing credentials.

Brave negotiation mode refers to a mode of negotiations where an opponent (client or the ENS) discloses their own credentials without any policy-driven access control bypassing the decision making part. In our example, when a server requests an administrator credential from a client, the client-side proxy will grant disclosure of that credential without access control on that credential. This mode has practical implications in cases where a client does not need fine-grained credential control but trusts the remote domain identity as sufficient for disclosing their own credentials.

7. Security Architecture Implementation

First achievement of the proxy implementation is enabling secure communications transparent to OKKAM-enabled applications. The proxy component implements all remote ENS APIs as locally deployed allowing easy and intuitive integration with ENS applications. The proxy provides local replicas of all ENS APIs (signatures). Second achievement is encapsulation of authentication and access control mechanisms for accessing ENS APIs by enforcing completely transparency to applications security-related processes. The client-side proxy automatically negotiates with the server-side proxy (the PDP service) for more access rights necessary to get service access.

The ENS provides all functionalities via Web Services APIs divided into two sets: *public non-secure* WS APIs[8] and *public secured* WS APIs[9]. The set of non-secure APIs contains those APIs

[8] `http://api.okkam.org/okkam-core/WebServices?wsdl`
[9] `http://api.okkam.org/okkam-core/WebServicesSecured?wsdl`

available for open and uncontrolled access (e.g., all query-related APIs) accessible by standard Web Services means.

The public secured APIs require secure and confidential communications with appropriate access rights for their execution. An important security aspect here is that all non-secure APIs are also provided as secured APIs (without access control but over a secure communication channel). In that way, a user can enforce privacy and confidentiality on all interactions with the ENS APIs including the non-secured APIs. For example, when a user wants to avoid any intermediary third parties monitoring what he queries to ENS.

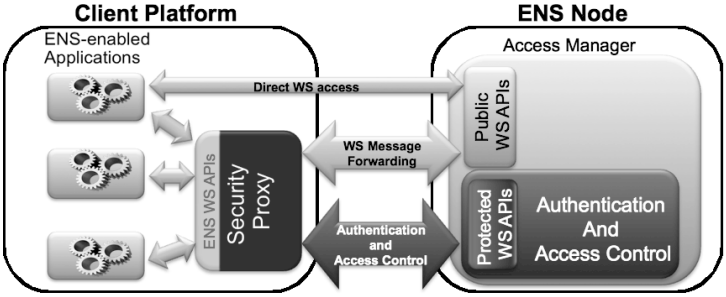


Figure 5: API-level Proxy Communications

All interactions with the secured ENS APIs are performed via the security proxy. Figure 5 shows the API-level message flow of proxy-based communications. Applications can establish directly or indirectly via proxy, communications for the non-secured APIs. The proxy component replicates all ENS APIs, both non-secured and secured ones, as locally accessible to applications in order to make access to those API as transparent as possible for application developers.

Another goal is to provide developers with uniform API access allowing all APIs to be accessed via the proxy component without keeping in mind which of those require secured access and which do not. The proxy component is aware of which APIs are secured and for those requiring no protection the proxy just dispatches the respective requests/responses to the public ENS APIs, as shown in the figure above.

When the proxy is configured to enforce secure communications on all interactions with the ENS, it will dispatch all APIs invocations to the secured versions of the APIs even if a non-secured API is locally invoked.

The proxy component has two main modes of usage: as a localhost service and as a library. The local host service implements the same remote WS APIs (interfaces only) but accessible on localhost connection[10]. This is an important mode addressing the needs of so-called "thin" applications, such as plug-ins for third party applications, or applications that cannot use the proxy as a library. In such case, an application invokes the WS APIs on localhost for accessing the remote WS APIs. The library usage, in contrast, provides the remote ENS APIs as local library API signatures for accessing the respective remote WS APIs.

7.1. Proxy Architecture Inside

The main goal of the proxy design is to establish secure and trusted end-to-end (beyond point-to-point) message-level communications traversing local area network firewall and remote (http) load balancer.

Figure 6 shows the proxy architecture and its insight view of how main components communicate with each other. The access control and trust negotiation component represent logic

[10]Client-side proxy component does not have to be run on a localhost but could be also run on a separate machine. In that case, one needs to additionally secure all interactions between applications and the proxy.

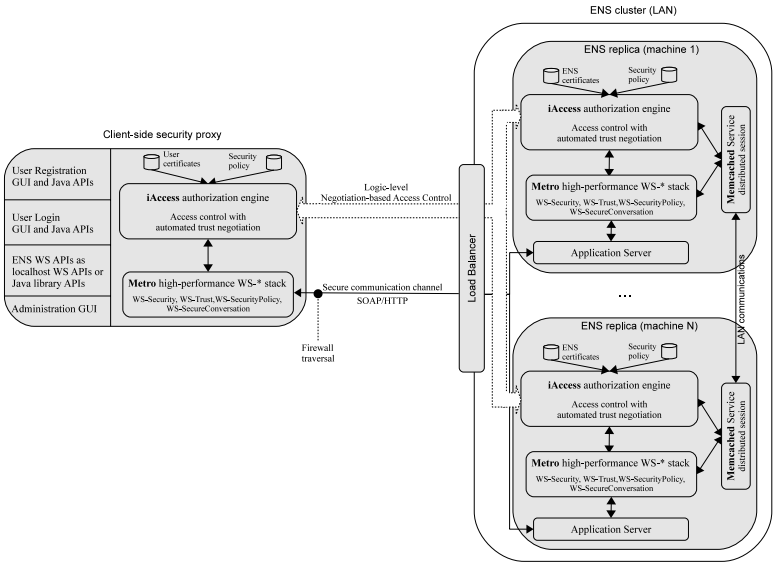


Figure 6: Proxy Architecture and Communications

expression of how trust is managed/established with potential ENS nodes and potential ENS users, while the WS security standards component is the lower level definition of how the trust requirements are communicated to an opponent. The last component provides the actual WS communication mechanism over a secure communication channel. The security channel provides end-to-end message communication, i.e. between a client and an ENS replica. The load balancer does not decrypt any messages to serve its functionality.

The trust negotiation based access control, presented in Section 6, has been implemented as a new release of iAccess system[11]. An important technical feature of the new iAccess release is the implementation of a service (daemon) running locally on a node replica which manages externally to a Java application server's executions of the underlying DLV[12] system thereby avoiding the related memory allocation issue by the OS.

Another novelty of the proxy component is the proper integration of iAccess with Metro[13], which provides state-of-the-art implementation of WS security standards. The effort dedicated to integrating the access control and negotiation component with the Metro component led to achieving an efficient and powerful end-to-end authorization process for Web services.

In addition to security specification and enforcement, the proxy component has been designed to provide two more functionalities: secure user registration to the ENS and secure administration of user certificates. The official OKKAM registration front-end[14] internally uses the proxy to perform secure user registration to the ENS. There is a dedicated GUI on the client-side proxy that allows secure and authorized administration of users. This functionality is important for the well-being of the ENS allowing authorized execution of several administrative tasks.

Server-side proxy scalability. One of the technical challenges in developing the ENS security mechanism is addressing scalability of security services having the same scalability as the ENS itself. Instead of having a separate server, inside the ENS cluster, handling all security communications we replicate all security functionality on any server in a cluster hosting the ENS

[11] `http://www.interactiveaccess.org`
[12] DLV is used for the underlying logic computations of the access control model [6].
[13] `https://metro.dev.java.net`
[14] `http://register.okkam.org`

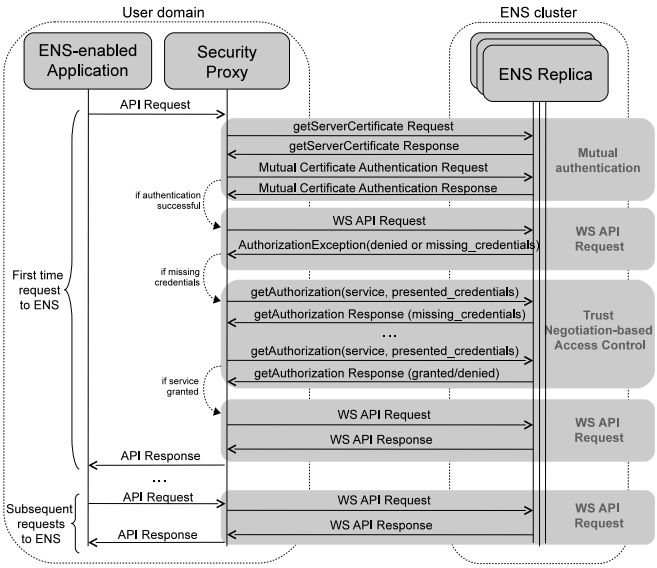


Figure 7: Proxy Interactions with the ENS

without imposing any sticky session requirement on the load balancer. In such a design choice, establishing secure and trusted communication to ENS inherits the scalability of ENS cluster.

We deployed the memcached[15] service for distributed session sharing between ENS replicas, as shown in Figure 6, achieving coherent decision making on any replica in a cluster. The access control module has been extended to use the memcached service for sharing access control session data. The Metro component has also been extended to use the memcached service for low-level (session-aware) encryption/decryption of incoming/outgoing messages.

Any replica of security services in a cluster replicates the same security policies, the same certificate authority used for issuing user certificates, and the same public-key certificate identifying a current replica node. Appendix A presents the designed PKI model.

7.2. Proxy Interactions with the ENS

A design goal behind the security proxy is to enable efficient security access to ENS. We have designed the security proxy component establishing single security session for multiple user requests regardless of which ENS replica handles the requests.

Figure 7 shows the main phases a client-side proxy component performs upon a request by an application. The figure shows the proxy interactions when a user application accesses an ENS API via the proxy for the first time. Any subsequent request is granted if a user has already presented (in previous interactions within the security session) enough credentials granting access to the request. Similarly, the user-side proxy bypasses all negotiation requests for credentials the server has already presented in previous interactions within the session.

Mutual Certificates security mechanism[16] is used for establishing secure and confidential message channel between users' applications and the ENS. Both client and server use X.509 certificates, issued by an OKKAM certificate authority, to authenticate each other. Once mutual authentication is established, there is a mandatory access control process if a registered user is allowed to access the requested ENS API. When a client needs more access rights for a given request an automated trust negotiation process is triggered for the missing credentials. Upon

[15] `http://memcached.org`

[16] Defined by WS-SecurityPolicy and WS-SecureConversation standards.

receiving the very first request the proxy first obtains the server certificate, verifies if it is trusted, and runs the mutual certificate security protocol establishing a session encryption key. All subsequent proxy interactions run over the already established confidential channel.

We gave the sever-side proxy component *cache* access decisions within a given security session so that in every repeated API request, the policy enforcement component bypasses the policy decision point if the service has been already granted within the session.

When the requested API is a non-protected API the proxy invokes the corresponding ENS API in the public space without using the security channel. Otherwise, the proxy invokes the API under the protected ENS API services using the already established security channel. When there are not enough access rights the ENS denies access to the requested API including a list of missing credentials. If missing credentials are required, the proxy triggers a negotiation process with the PDP API. This phase may require more interactions between the client-side proxy and the PDP depending on the underlying security policies.

Upon successful negotiation the client-side proxy again invokes the WS API and forwards the response back to the application. If at this step the server (for whatever reason) returns an authorization exception for missing credentials the user-side proxy does not initiate a new negotiation process, but instead returns the authorization exception (of access denied) back to the application. All subsequent API requests to the ENS share the same security session without repeating trust negotiation on already negotiated credentials.

7.3. Certificate Revocation

Certificate revocation service facilitates the provision of revocation and certification revocation status service in establishing trusted communications. Certificate revocation is based on certificate revocation lists[17] (CRLs) at ENS side. CRLs are maintained within the ENS cluster by means of the persistent storage functionality provided by the underlying database module of unified data storage and retrieval within a cluster. There are two certificate revocation lists maintained: *okkamall.crl*[18] and *okkamauthorities.crl*[19]. The first CRL regards revocation certificates of all OKKAM entities both end users and OKKAM authorities, while the second CRL regards revocation certificates of OKKAM authorities only. The okkamall CRL is to be used primarily by entities requiring validation of end users such as the ENS servers, the administration console and the ENS Web toolkit, while the okkamauthorities CRL is to be used primarily by end users when interacting with OKKAM front-ends or with the ENS.

The okkamauthorities CRL is a subset of the okkamall CRL. Furthermore, okkamauthorities is expected to have substantially fewer revocation certificate entries with regards to the okkamall CRL, as well as, having much less frequency of certificate revocation. The motivation for having okkamauthorities is to make efficient end users (the most delicate use case) certificate revocation status control when interacting with the ENS and with the ENS front-ends.

There is dedicated ENS API providing the most up-to-date response of certificate revocation status compared to the CRL approach. The API serves solely the security proxy functionality. It directly uses the persistent storage data (where each revoked certificate has a single entry in the storage) enabling efficient response. There is a dedicated GUI of the security proxy facilitating management of CRLs on ENS where access to respective APIs of certificate revocation is controlled to users (for user revocation) or administrators (forced revocation).

7.4. End Users Authentication via OKKAM Web Front-ends

User communications with the OKKAM Web front-ends are secured by means of HTTPS[20] protocol where end users register with the ENS to obtaining a certificate attesting to them

[17] `http://tools.ietf.org/html/rfc3280`
[18] `http://api.okkam.org/okkam-core/crl/okkamall.crl`
[19] `http://api.okkam.org/okkam-core/crl/okkamauthorities.crl`
[20] `http://tools.ietf.org/html/rfc2818`

being OKKAM users. The security proxy has its own certificates in order to be authorized to access ENS, such as, a public-key certificate of being legal OKKAM entity, trusted entity creator attribute certificate necessary for creating entities on behalf of end users, or administrator attribute certificate for accessing ENS life cycle operations such as merge, split, delete.

In all cases, the proxy component authorizes to the ENS with its own credentials and whenever required by the ENS APIs presents end user certificate. There are two main front-end applications: entity-creator front-end and administration console front-end. Both of them run their own security proxy component with their own certificates.

Entity creator[21] helps users to create ENS entities. When the proxy is run by the entity creator Web front-end, the proxy needs a registration certificate of being an OKKAM registered entity, and a trusted entity creator attribute certificate to be authorized to create entities on *behalf* of end users. There are dedicated ENS APIs, different from the create entity APIs, that allow ENS entity creation on behalf of end users. The new APIs require an additional input parameter, with respect to the standard create entity APIs, the end user public-key certificate on behalf of which the entity creator acts. By using these APIs the ENS marks (logs) the end user as the author of an entity creation act, and stores the identity of the entity creator service as the intermediary entity the user created the ENS entity.

Administration console[22] is another important front-end application facilitating the administration of ENS entities. Access to this front-end is granted only to users that have been approved to act as administrators. The approval process requires that a registered OKKAM user sends its public-key certificate to an ENS administrator for approval and the inclusion of that certificate in a set of recognized administrators by the administration front-end application. The administration console is accessible only via HTTPS with mutual client-and-server certificate authentication. There is access control if the user certificate (obtained from HTTPS protocol context) is among the approved users of acting as administrators.

Similarly to the entity creator use case, all entity life cycle APIs have alternative APIs providing the same operations but on behalf of end users with an additional argument: the end user public-key certificate. The ENS logs, as the author of a current life cycle operation, the end user acting via the console interface. The security proxy of the administration console is authorized as being a legal OKKAM entity and, secondly, of having (own) administration privileges, in order to invoke life cycle operations on behalf of end users.

Private ENS node communications with the ENS are treated as communications by a normal OKKAM registered user. In this use case, the private ENS node is not considered to act on behalf of end users (although a private ENS does offer ENS services to domain/company-specific end users) and it is authorized to perform secure access to ENS as is the case of a registered user. The certificate attribute model of the ENS allows private nodes to be recognized by their private node attribute certificate and be given some privileged access. However, this feature is left open for the future evolution of ENS.

8. Security Architecture Performance Evaluation

We will present the performance of the security proxy and the PDP service of ENS. All measurements have been performed on both a locally installed ENS core and on the real ENS platform[23].

8.1. Security Proxy Performance

There are five distinguishable phases of ENS proxy-based security communications in a defined order:

[21] `http://api.okkam.org/EnsWebToolKit`
[22] `http://api.okkam.org/AdministrationConsole`
[23] ENS platform accessible at `http://api.okkam.org`

1. Get server certificate for secure channel establishment (non-protected access).
2. Secure communication channel establishment using client and server certificates.
3. Trust negotiation process in *brave mode* for establishing administrator privileged access.
4. Trust negotiation process in *cautious mode* for establishing administrator privileged access.
5. API invocation over a secure channel with grant access decision (enough credentials).
6. API invocation over a secure channel with cached access decision.

Figure 8 reports the first set of experiments with total round-trip messages as measured on a client side. The figure shows both sets of results alongside each other: left-hand side columns denote security performance on localhost tests and right-hand side columns security performance on the ENS platform. The localhost experiments show the security performance with almost no communication cost, while the measurements with the ENS platform show results with the real communication overhead.

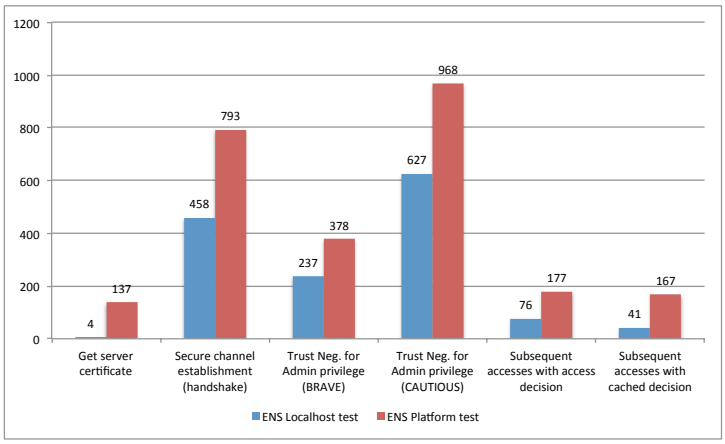


Figure 8: ENS localhost and ENS platform tests performance (milliseconds)

Each number in this first set of tests is reported as an average of 50 trials run sequentially. The interesting part of this performance comparison is the sensitivity of security process with respect to communication costs instead of the expected security computational costs for secure API invocation case (points 5 and 6 above).

Upon a service request via the security proxy, a client will experience a total of 970 ms of secure API invocation[24] excluding the time of actual service execution. The 970 ms time splits in 793 ms of secure handshake establishment and 177 ms of secure API request. Any repeated access to the same API is measured to 167 ms including the security performance for entire message communication (request/response) and with cached access decision. In the case the requested service requires access by administrators, a client will experience a total of 1.348 seconds (793 ms of secure establishment + 378 ms TN of admin in brave mode + 177 ms secure access with decision), and will experience 167 ms for any repeated access request to that service. Get server certificate time is not counted in the above examples as it can be performed once prior to any session establishment by the proxy.

ENS platform test. The ENS platform is dedicated by the project consortium for the real production services [26]. The platform has the following hardware:

- Machine 1 (of cluster): 8 core Intel(R) Xeon(R) CPU E5420 2.50GHz, 8GB of DDR2 SDRAM 667 MHz, Linux (Debian) OS.

[24] For example in the case of creating a new entity service accessed by a registered user, refer to Appendix A.

- Machine 2 (of cluster): 8 core Intel(R) Xeon(R) CPU E5420 2.50GHz, 16GB of DDR2 SDRAM 667 MHz, Linux (Debian) OS.

The client machine for these tests is the machine used for the localhost experiment with the same client-side security proxy and the application using the proxy to access the ENS platform. We used the university's Internet connection[25]. Figure 8 reports the ENS platform performance results of the different phases of proxy-based security.

ENS local test. We have installed and run an ENS core on a machine with the following parameters: Mac OS X, CPU Intel Core2 Duo at 2.5GHz, RAM 4GB DDR3 at 1GHz. On the same machine we also run the proxy component with a client application using the proxy to access ENS APIs via the localhost connection. It is important for this experiment to measure (client observable) security performance with no communication cost. In other words, we measured security-related computations including client side encryption/decryption, server-side encryption/decryption and access decision time.

8.2. Bulk Import Security Performance and Scalability

In the following we present the ENS platform performance for the use case of bulk import – automated tool support for entity import (creation) from external data set – considered as the most sensitive use case to security overhead. The goal is to show how total security overhead (including secure communications establishment/handshake, encryption/decryption, access control, and communication overhead) impacts on bulk import functionality.

Sequential Access. We conducted a set of experiments with the ENS platform to measure the average access time on performing 1000 times sequential access with and without security. Table 2 shows the average time per single API access, client observable round-trip time including communication cost. We used a dedicated API in the pubic ENS space and the same in the protected ENS space with controlled access to authenticated users. The average security overhead for accessing the API was measured to **26** ms, much below the identified performance requirement of few hundreds of a second (Section 4).

	Non-secured access	Secured access	Security overhead
Average request and response time of API access	142 ms	168 ms	**26** ms

Table 2: Bulk import sequential access average time

Concurrent Access. The next set of tests is used to measure the scalability of the security solution in the context of concurrent service requests to the ENS. We transformed the bulk import sequential test into a concurrent access test where we defined several client applications each running its own security proxy. We performed a bulk import test having N number of clients running concurrently each in different security session making 10 requests to a protected service on the ENS. We repeated the concurrent test for a non-protected service access to derive the security overhead. We used two computers with the same hardware configuration as that of the aforementioned ENS localhost test each of them running concurrently N/2 number of clients.

Figure 9 shows client perceived (round-trip) access time average of M number service requests performed by N number of concurrent clients. The security overhead increases together with the increase of concurrent service requests. If we look at the case of 100 concurrent clients with a total of 1000 service requests we have a security overhead of about 250 ms which is still within the identified requirement of acceptable security overhead. Given the last case of 200 concurrent clients with 2000 requests the security overhead is 469 ms, not much above the identified requirement. Comparing the numbers of both tests, we can see that up to 40 clients

[25] University of Malaga (Spain).

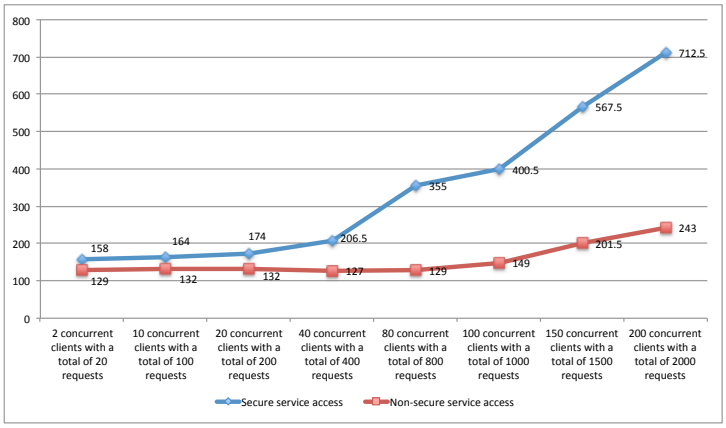


Figure 9: Bulk import concurrent access average time (milliseconds)

with 400 requests in the concurrent access test the security overhead remains comparable to that of the sequential access test where the difference is due to fluctuations in the network connection.

8.3. PDP Performance

An important part of the whole set of experiments is the security performance of the PDP service. Measurements on this level of granularity show the actual capacity of ENS single-core instance in response to secure versus non-secure service access.

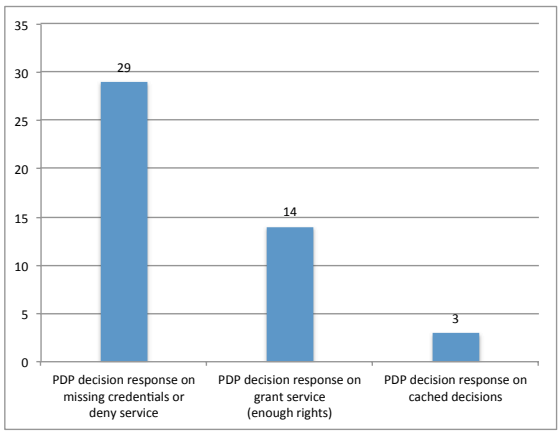


Figure 10: PDP performance per request and per single node (milliseconds)

Figure 10 shows the PDP decision making on the different access control cases obtained as part of the ENS platform test on machine 2 of the cluster. The PDP response time ranges from 29 ms down to 3 ms per single request. The time of 29 ms is the PDP security response time for a single negotiation step. The overhead is mostly influenced by the access decision process of computing missing credentials. Inherent to the adopted access control model is that the deny access decision is taken when no missing credentials are found for a potential negotiation[26]. The ENS security overhead of granting access to a service takes 14 ms. This case happens either

[26]If a user does not have enough access rights to be granted a service request and no missing credentials are found that would grant the user access to that service, the server denies access.

after a successful negotiation (has taken place) or if a client requests a service within an existing session for which he has enough rights but the access decision is not cached.

The PDP security overhead with cached access decisions, the case in every repeating service call within the same security session, is measured 3 ms overhead to an API execution. Essentially, this time includes only the access times to memcached service for session data retrieval for cached decisions. The cached access decision time corresponds to the most frequent case of using ENS APIs, especially, for the bulk import scenario. The ENS security overhead with cached access decision is measured to 11 ms where 8 ms time is taken by Metro-level cryptographic operations.

9. Discussion

Security Management. The concept of proxy-based security facilitates transparent to end users (business-driven) security management. This allows users and companies to define security requirements on communications with the ENS without imposing any changes on applications to conform to new security settings. For example, if a pharmaceutical company wants to enforce confidential access to the company's private ENS (with identifiers of pharmaceutical products), the company will have to set all APIs as protected APIs. In such a case, the proxy will enable all end users' (company's) applications communicate with the ENS APIs confidentially and conform to the respective access control requirements without any application-level changes. End users could also enforce confidential access to all ENS APIs transparently to application-level by configuring client-side proxy forward all API requests to their protected versions.

The security model behind the proxy allows user-centric control of credential disclosure. There is a credential policy on the user side that governs which user's credentials are to be used and under what requirements. In such a case, a user of a company can restrict his credentials only with the company's ENS, and OKKAM specific credentials with the public ENS, respectively.

System Robustness. Another aspect towards system adoption is its robustness against possible threats. Although system robustness may cover many system aspects beyond current paper's scope, we will overview the threat closely relevant to the system robustness against denial of service (DoS) attacks on the negotiation-based access control service (the PDP service).

Trust negotiation provides systems for open environments (such as the ENS) with a protection mechanism from strangers and gradual trust establishment with previously unknown entities. However, stateful negotiation implies active server behavior with client agents (such as, asynchronous negotiation with WS-Notification [23]). This aspect impacts significantly on potential DoS attacks on the system making clients explicitly keep servers in negotiation by making servers dedicate computational and memory resources for negotiation state maintenance to interact with clients.

Our stateless server-side negotiation focuses, shifts the negotiation logic on the policy evaluation level without reasoning on higher-level negotiation states. For example, a server agent of stateless negotiation does not need to reason for situations of negotiation deadlock, what would be the case of statefull negotiation, because the server agent is passive (stateless) of previous negotiation states. The client agent, in turn, keeps stateful negotiation (client-driven) and is responsible for the cooperative execution of negotiation steps and completion of the negotiation process. We concluded that in large scale intensive systems stateless server-side negotiation enhances system robustness against DoS attacks more than stateful negotiation approaches.

We summarize the technical decisions taken to further enahnce system robustness against DoS attacks: first, replicating the PDP service throughout the nodes in a cluster ensuring a certain level of service availability, second, enhancing negotiation protocol behavior by keeping a record of requested credentials from clients (Section 6.1) within a session, and, third, defining an upper bound of interaction steps a client is permitted for successful negotiation on any service access (by analyzing the server security policies) so that the server timely terminates any negotiation process exceeding the upper bound, for example, if an attacker intentionally enters into a negotiation deadlock.

Intrusion Detection. Towards a more complete security protection of ENS is the adoption of an Intrusion Detection System (IDS) [31] as part of the security framework. An IDS consists of (software) components that monitor actions executed by users or applications. There are two kinds of IDS: those which use signatures to detect attacks whose behavior is well understood (e.g., [32]) and those which use statistical or data mining analysis (e.g., [33]) to detect attacks.

A potential synergy of integration of an IDS into the OKKAM security framework is that the IDS behavioral model could be enhanced by taking into account several factors from the access control process, such as user behavior and credentials obtained during the negotiation process, for a more advanced reasoning on intrusion detection.

10. Conclusions and Future Work

The paper has presented a security framework for service provisioning of an ENS – a reliable data intensive load-balancing cluster system. There are three main foundational elements underpinning OKKAM security infrastructure: (i) Digital certificates qualifying OKKAM community of users, their privileges, and OKKAM infrastructure nodes. (ii) Trust negotiation-based access control enabling on-the-fly automated credential establishment. The negotiation protocol improves on previous work to provide stateless server-side negotiation which scales well to the needs of the ENS. (iii) Proxy component engineering providing security, trust and privacy control aspects inside a transparent configuration-only component with high-level security abstraction.

A detailed security performance evaluation has been presented supporting conclusions of scalable security design and efficient implementation with respect to the performance guidelines set out by the project consortium.

Future Work. The large-scale ENS repository introduces new challenges related to specification and management of policy requirements for the vast number of repository entries. Future work will move in the direction of extending the negotiation based access control with the semantic access control (SAC) model [34, 35]. The SAC model defines in a scalable and flexible way semantic abstraction of access policy specification from policy applicability to resources. The goal is to qualify a large amount of ENS repository entities to relatively few access policies and, as consequence, facilitate scalable policy management and enforcement.

Implementation- and performance-wise direction of future work is to provide the underlying logic computations of deduction and abduction as Java-based solutions, avoiding the execution of an external to Java DLV system. We plan to approach this aspect by adopting a Java-based Prolog inference engine supporting both reasonings [36].

Acknowledgements

This work is partially supported by the 7th Framework Programme of the European Commission under the project **OKKAM** (Enabling a Web of Entities, contract ICT-215032); partially supported by the Spanish national project **ConTur** (AVANZAIDI-AAP-090521, Plataforma para la Gestión Inteligente de Contenidos en el Ámbito del Turismo); and partially supported by the project **DESEOS** (TIC-4257, Dispositivos Electrónicos Seguros para la Educación, el Ocio y la Socialización) by the government of Andalucía.

References

[1] P. Bouquet, H. Stoermer, C. Niederee, A. Maña, Entity Name System: The Backbone of an Open and Scalable Web of Data, in: Proceedings of the IEEE International Conference on Semantic Computing, (ICSC 2008), IEEE Computer Society, 2008, pp. 554–561.

[2] Z. Miklós, N. Bonvin, P. Bouquet, M. Catasta, D. Cordioli, P. Fankhauser, J. Gaugaz, E. Ioannou, H. Koshutanski, A. Maña, C. Niederée, T. Palpanas, H. Stoermer, From web data to entities and back, in: Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10), LNCS, Springer, 2010.

[3] P. Bouquet, H. Stoermer, B. Bazzanella, An Entity Name System (ENS) for the Semantic Web, in: Proceedings of The Semantic Web: Research and Applications (ESWC2008), Vol. 5021/2008 of LNCS, Springer Berlin/Heidelberg, 2008, pp. 258–272.

[4] K. Seamons, W. Winsborough, Automated trust negotiation, US Patent and Trademark Office. IBM Corporation, patent application filed March 7, 2000 (2002).

[5] M. Winslett, T. Yu, K. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, L. Yu, Negotiating trust in the Web, IEEE Internet Computing 6 (6) (2002) 30–37.

[6] H. Koshutanski, F. Massacci, Interactive access control for autonomic systems: from theory to implementation, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 3 (3).

[7] H. Koshutanski, F. Massacci, A negotiation scheme for access rights establishment in autonomic communication, Journal of Network and System Management 15 (1).

[8] T. Y. C. Woo, S. S. Lam, A framework for distributed authorization, in: Proceedings of the 1st ACM conference on Computer and communications security, 1993, pp. 112–118.

[9] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, A. Essiari, Certificate-based access control for widely distributed resources, in: Proceedings of Eighth USENIX Security Symposium (Security'99), 1999, pp. 215–228.

[10] M. Zurko, R. Simon, T. Sanfilippo, A user-centered, modular authorization service built on an RBAC foundation, in: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Press, 1999, pp. 57–71.

[11] D. W. Chadwick, A. Otenko, The PERMIS X.509 role-based privilege management infrastructure, in: Seventh ACM Symposium on Access Control Models and Technologies, ACM Press, 2002, pp. 135–140.

[12] W. Winsborough, K. Seamons, V. Jones, Automated trust negotiation, in: Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX), Vol. 1, IEEE Press, 2000, pp. 88–102.

[13] M. Winslett, An introduction to trust negotiation, in: First International Conference on Trust Management (iTrust'03), Vol. 2692 of LNCS, Springer, 2003, pp. 275–283.

[14] T. Yu, X. Ma, M. Winslett, Prunes: an efficient and complete strategy for automated trust negotiation over the internet, in: Proceedings of the 7th ACM conference on Computer and communications security (CCS '00), ACM, 2000, pp. 210–219.

[15] K. Seamons, M. Winslett, T. Yu, Limiting the disclosure of access control policies during automated trust negotiation, in: Proceedings of the Network and Distributed System Security Symposium, 2001.

[16] T. Yu, M. Winslett, K. E. Seamons, Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation, ACM Transactions on Information and System Security (TISSEC) 6 (1) (2003) 1–42.

[17] A. Squicciarini, E. Bertino, E. Ferrari, F. Paci, B. Thuraisingham, PP-trust-X: A system for privacy preserving trust negotiations, ACM Trans. Inf. Syst. Secur. 10 (3) (2007) 12.

[18] A. J. Lee, M. Winslett, K. J. Perano, TrustBuilder2: A reconfigurable framework for trust negotiation, in: Proceedings of the 3rd IFIP WG 11.11 International Conference on Trust Management, Springer, 2009.

[19] P. A. Bonatti, D. Olmedilla, Driving and monitoring provisional trust negotiation with metapolicies, in: Proceedings of the 6th IEEE Workshop on Policies for Distributed Systems and Networks (POLICY-05), IEEE Computer Society, 2005, pp. 14–23.

[20] E. Bertino, E. Ferrari, A. C. Squicciarini, Trust-X: A peer-to-peer framework for trust establishment, IEEE Trans. on Knowledge and Data Engineering 16 (7) (2004) 827–842.

[21] A. J. Lee, M. Winslett, J. Basney, V. Welch, The traust authorization service, ACM Transactions on Information and System Security (TISSEC) 11 (1) (2008) 1–33.

[22] H. Koshutanski, A. Lazouski, F. Martinelli, P. Mori, Enhancing grid security by fine-grained behavioral control and negotiation-based authorization, International Journal of Information Security 8 (4) (2009) 291–314.

[23] I. Constandache, D. Olmedilla, F. Siebenlist, Policy-driven negotiation for authorization in the grid, in: Proceedings of 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07), IEEE Computer Society, 2007, pp. 211–220.

[24] W. Nejdl, D. Olmedilla, M. Winslett, PeerTrust: Automated trust negotiation for peers on the semantic web, in: VLDB Workshop on Secure Data Management (SDM), Vol. 3178 of Lecture Notes in Computer Science, Springer, 2004, pp. 118–132.

[25] OKKAM Project Consortium, Integration Prototype OKKAM Infrastructure V2, project deliverable document D5.4 available at `http://www.okkam.org/deliverables` (2009).

[26] OKKAM Project Consortium, Integration Prototype OKKAM Infrastructure V3, project deliverable document D5.6 available at `http://www.okkam.org/deliverables` (July 2010).

[27] OKKAM Project Consortium, OKKAM in Numbers V4, OKKAM project deliverable document D12.7 available at `http://www.okkam.org/deliverables` (September 2010).

[28] X.509, The directory: Public-key and attribute certificate frameworks, ITU-T Recommendation X.509:2005 | ISO/IEC 9594-8:2005 (2005).

[29] SAML, Security Assertion Markup Language (SAML), `http://saml.xml.org/saml-specifications`.

[30] Ó. Cánovas, M. Sánchez, G. López, R. del Campo, S. Neinert, J. Rauschenbach, I. Thomson, Architecture for unified SSO, deliverable DJ5.3.2, available at `http://www.geant2.net/alice/upload/pdf/GN2-08-080v2-DJ5-3-2_Architecture_for_Unified_SSO.pdf` (May 2008).

[31] T. F. Lunt, A survey of intrusion detection techniques, Computers and Security 12 (4) (1993) 405–418.

[32] U. Lindqvist, P. Porras, Detecting computer and network misuse through the production-based expert system toolset (P-BEST), in: In Proceedings of the IEEE Symposium on Security and Privacy, 1999, pp. 146–161.

[33] D. Barbará, J. Couto, S. Jajodia, L. Popyack, N. Wu, ADAM: Detecting intrusions by data mining, in: In Proceedings of the IEEE Workshop on Information Assurance and Security, 2001, pp. 11–16.

[34] M. Yagüe, M. Gallardo, A. Maña, Semantic access control model: A formal specification, in: Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS'05), Springer, 2005, pp. 24–43.

[35] M. Yagüe, A. Maña, J. López, A metadata-based access control model for web services, Internet Research 15 (1) (2005) 99–116.

[36] H. Christiansen, V. Dahl, HYPROLOG: A new logic programming language with assumptions and abduction, in: Proceedings of the 21st International Conference on Logic Programming (ICLP 2005), Vol. 3668 of LNCS, Springer, 2005, pp. 159–173.

[37] H. Koshutanski, A. Maña, A semantic approach to access control and automated credential negotiation for decentralized online repositories / an OKKAM project use case, in: Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008), ISSN 1613-0073, CEUR Workshop Proceedings, Rome, Italy, 2008.

Appendix A. OKKAM Certification and Attribute Management

Figure A.11 shows the public key and privilege management infrastructure (PKI/PMI) model designed and deployed for the OKKAM community.

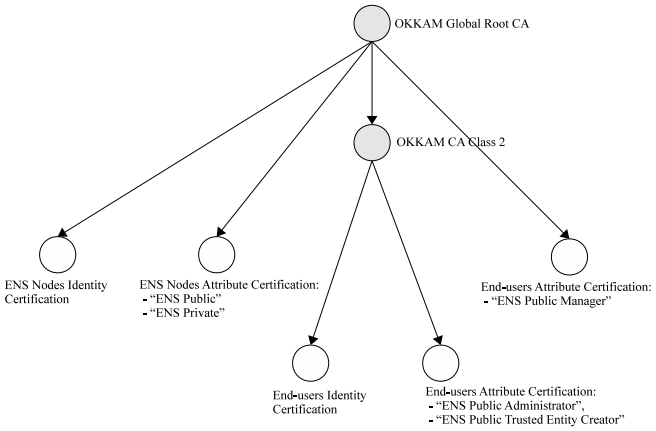


Figure A.11: OKKAM certificate infrastructure

The OKKAM Global Root CA is the authority trust starts from. It can be seen as the representative of the project consortium. The root is the most sensitive authority, which sets up trust in OKKAM infrastructure and in OKKAM community. In order to reduce the risk of compromising the global root and for the sake of better management, the OKKAM Global Root CA delegates to a subordinate CA, called OKKAM CA Class 2, the responsibility to certify all users and entities of the OKKAM community.

In addition, the OKKAM Global Root CA has been defined with two more responsibilities: (i) certifying ENS nodes by means of identity and attribute certificates such as ENS Public and ENS private nodes[27], and (ii) certifying/promoting ENS Public managers. The former case allows the OKKAM consortium to keep control on what ENS systems are legally certified as ENS systems so that any third party can verify if it trusts that node as being part of the OKKAM infrastructure. When the OKKAM Root CA certifies a private ENS node this means that the private ENS node is part of the OKKAM community.

[27] The OKKAM Public Key Infrastructure design supports also the use case where the OKKAM Global Root CA delegates to a subordinate CA responsibility to certify ENS nodes as being part of OKKAM community.

Attribute name	Permissions on the ENS
ENS Public Manager	– High-level permissions on certificate management. – No permissions on entity creation, update, and life-cycle operations
ENS Public Administrator	– Limited permissions on certificate management. – High-level permissions on entity creation, update, and life-cycle operations (such as merge, split, delete ENS entities) including permissions on behalf of end users.
ENS Public Trusted Entity Creator	– Permissions on create new ENS entities on behalf of end users. – Permissions on update ENS entities on behalf of end users.
OKKAM Registered User	– Permissions on create new ENS entities. – Permissions on update ENS entities by append new attributes only.

Table A.3: Certifiable attributes and permissions

The OKKAM consortium is represented by ENS public managers certified by the OKKAM Global Root CA. The goal is to allow the OKKAM consortium delegate responsibility of managing the ENS to dedicated people. ENS public managers are the principle entities that can approve (delegate specific responsibility) to other key entities, such as administrators of ENS, trusted entity creators, etc. for the sake of well-being of the ENS.

Table A.3 summarizes the permissions for each of the attributes. All certificate management aspects have been provided as dedicated ENS APIs and accessed via the security proxy administration GUI. The goal of the above management of responsibilities is to better scale to the expected ENS community expansion of users.

An important aspect of the ENS Public Manager attribute is that it has no permissions allowed on entity creation and life cycle management. The main reason is to separate the responsibility of this attribute from the other attributes, especially, from the administrator privileges. However, a public manager can self-promote himself to become ENS administrator and thus obtaining the high-level permissions on ENS operations. In this case, the access control policy will force him to use only one of the attributes at a time, that is forcing a separation of duty constraint on the presence of those attributes. To face scalability and flexibility when OKKAM community grows, a user promoted as ENS administrator is given the right to promote registered users of being ENS trusted entity creators.

Achieving scalable access to ENS services from external domains with own PKI models and certifiable attributes could be approached by semantic interoperability of certificates [37].

User registration. Users register via an OKKAM front-end for obtaining a public-key certificate of being OKKAM users. Once registered, users can be promoted to take higher level of permissions on ENS via the certificate management process above. A user obtains its registration data in a single file encrypted with the password the user specified during registration. The registration file conforms to PKCS12[28] standard. The registration data contains comprehensive data structure allowing end users establish trust with other registered OKKAM entities.

Users install their certificate data in a Web browser, OS, or security proxy specifying username/password of registration so that the system extracts the certificate information used for the actual user authentication with remote sites. Users' attribute certificates are installed in the security proxy but their usage requires the user initialize the security proxy with a public-key certificate matching the identity information of the attribute certificates.

Attribute certification. Obtaining any attribute certificate is a non-automated process involving an approval by either an ENS public administrator or an ENS public manager. An OKKAM registered user contacts any of the entities to get an approval by sending its public-key certificate. Upon approval, the administrator or the manager uses the security proxy (its GUI) to get authorized access to ENS certificate management APIs for user attribute certification. The ENS in turn e-mails the user its new attribute certificate.

[28] Personal Information Exchange Syntax Standard: `http://www.rsa.com/rsalabs/node.asp?id=2138`