A Kerberos security architecture for web services based instrumentation grids

Athanasios Moralis*, Vassiliki Pouli, Symeon Papavassiliou, Vasilis Maglaris

Network Management & Optimal Design Laboratory (NETMODE), National Technical University of Athens (NTUA) Athens, Greece

1. Introduction

A Grid is a large-scale generalized network system distributed across multiple organizations and administrative domains. In addition to legacy Grids that can be seen as a distributed batch processing platform, other grid environments have been emerging over the last few years. These grid environments enable users to remotely manage and control distributed resources and heterogeneous instruments, by providing standardized interfaces and unified work environments. Examples of such environments include Equipment Grids [1], Health Grids [2], Instrumentation Grids [3] and Control Grids [4]. In principle, their goal is to provide interactive near real time functionality to the Grid applications according to specific Quality of Service (QoS) requirements. Depending on the nature of the required QoS and the applications under consideration, several approaches that attempt to address the QoS constraints have been reported in the literature, that range from managing service level agreement contracts [5,6] and providing ways for efficient task management [7] and fault tolerance in grid systems [8], to directly reducing the system response time. The latter approach essentially aims at improving the responsiveness of the operations by reducing the delays in the system, which are mainly caused either by the network or the service processing on the application side.

Our objective in this paper is to design an efficient security architecture which authenticates and authorizes users or intermediate services that act on behalf of users to securely and reliably interact in near real time with services that monitor and control resources distributed across the network. The intent of the overall infrastructure is to support communication and interactions between two types of entities: Users and resources, while decreasing the delay required to authenticate and authorize the users that need to interact with the resources. The resources under consideration (e.g. instruments) are exposed as Web Services (WS), while the users control these resources by calling the corresponding operations of the web service.

Web Services provide open and interoperable standards for the creation and access of standardized interfaces, allowing users to manage distributed resources in a reliable and flexible way. They are based on XML encoded messages, communicating via the SOAP protocol, normally over HTTP. WSs allow for interactions in a TCP/IP network (Intranet or the Internet) enabling the creation of complex applications based on distributed entities.

It should be noted that the SOAP overhead on the message size degrades the performance significantly [9] compared to protocols that use binary messages like Java RMI or CORBA. Additionally, due to the message based nature of the SOAP protocol, each message in general is sent as a different TCP connection or HTTP session. Thus for each SOAP message, a different session needs to be established. This functionality, along with the required XML parsing of the message (serialization–deserialization) is very resource intensive in both computation and computer memory allocation. XML acceleration has been partially realized by offloading XML processing to dedicated appliances and hardware
devices. Their operation primitive is based on moving specific algorithms and functions to specifically designed hardware. Additionally, some of these appliances can perform XML routing and advanced security features, such as encryption, signatures that follow the WS-Security standards.

Security and near real-time control of the instrumentation Grid environments, usually spanning multiple domains, is a major and complex architectural challenge, elevating the importance of the corresponding security infrastructure [10,11].

Following the basic principles of Grids, the most important security requirements are:

- **Single Sign-on**: The user signs-in once and uses multiple resources (if authorized), without the need of multiple sign-ins for authentication.
- **Authorization to resources**: A user obtains access to a resource based on local policies and their enforcement.
- **Credential delegation**: The user delegates his credentials to intermediate services for them to act on his behalf.
- **Communication integrity**: Data exchanged should not be altered by an intermediary.
- **Communication confidentiality**: Data should not be disclosed to unauthorized entities.

1.1. Motivation, objectives and contribution

The security architecture introduced in this paper is designed to support the above security requirements of Instrumentation Grids. Our main goal is to improve security performance, maintaining at the same time interoperability with the Grid Security Infrastructure (GSI). Therefore, the proposed security architecture provides a WS compliant approach that covers all the basic security requirements presented above, while aims at preserving near real-time requirements of delay sensitive applications. It should be noted that the proposed architecture conforms to the guidelines of Open Grid Services Architecture (OGSA) [12] principles, as described later in the paper.

Our inspiration for the proposed security architecture has been the Grid enabled Remote Instrumentation with Distributed Control and Computation (GRIDCC) project [3] funded by the European Community, that aims at providing near-real-time and secure access to and control of distributed complex instrumentation. GRIDCC exploits grid opportunities for the secure and collaborative work of distributed teams, in order to remotely operate and monitor scientific equipment using the grid’s massive memory and computing resources for storing and processing data generated by this kind of equipment [13]. The successful operation of instruments distributed across the grid often requires rapid interaction with computing and/or storage resources and a large number of other instruments. This often calls for elevated delay and throughput requirements from all the components of the GRIDCC architecture, including the Security Services.

Among the critical and novel concepts introduced in GRIDCC architecture is the **Instrument Element** (IE) [14]. The IE is a set of services that provides the needed interface and implementation to enable the remote control and monitoring of physical instruments. Therefore, there is an elevated need for QoS regarding the response of the IE. This is essential for the interactivity of the application, which can be expressed in delay times between the service request and the service response. Smaller delay times (latency) translate to better experience thus fulfilling the near real time requirements of the application. Indicative applications include the **Synchrotron Radiation Storage Ring** at Elettra [15] as well as the **Run Control System** of the Data Acquisition System that belongs to the Compact Muon Solenoid (CMS) detector [16] situated at the **Large Hadron Collider** (LHC) at CERN. Such applications operate instruments or equipment distributed across the grid environments and therefore messages are exchanged across the network containing specific instrument operations such as: “turn (On/Off)”, “changeOrbit (Right/Left)”. When sending these messages the delay is comprised of a) the network delay and b) the processing delay of the message on the server side. The network delay depends on the network traffic, conditions and bandwidth, while the delay time at the service varies through the various applications because it depends on the processing of the message serialization, deserialization, the security processing and the processing of the actual service. Successful support of these applications usually requires delay/response times in the order of human reaction time (low latency), parallel control of 10^4 nodes/instrument accessed through the web, and fast enough information collection to allow monitoring by using aggregate collection at the order of 10^4 messages/s [14].

The delay-throughput requirements are dealt within our approach by exclusively using symmetric cryptography and the **Kerberos protocol** [17] to distribute the secret keys among the communicating parties. Asymmetric (Public Key) cryptography is not appropriate for adoption in delay sensitive grid applications, as its heavy overhead aggravates the low latency required. Our architecture utilizes **GSI X.509 Certificates** or **Proxy Certificates** (RFC3820) for the initial authentication of a user. However, it subsequently maps this identity to a Kerberos one and utilizes **WS Security Kerberos Token Profile** for embedding user credentials within WS exchange mechanisms. It then provides user authorization, thus realizing a complete AAI (Authentication & Authorization Infrastructure). In order to evaluate, understand and quantify the achievable performance improvements, we have compared the WS Security Kerberos Token Profile [18] of our approach against the X.509 Token Profile [19] that is applied as an authentication mechanism to secure the message exchanges when GSI certificates are used. As we further elaborate in the sequel, our comparative measurements demonstrated that the Kerberos Token Profile exhibits up to 50% message throughput improvement over the X.509 Token Profile, under high CPU load on the server. This improvement directly affects user response time of the service.

The rest of the paper is organized as follows: in Section 2 some related work is presented, while in Section 3 the Web Service Security specifications are summarized along with the two profiles to be compared (WS Security X.509 and Kerberos Token Profiles); in Section 4 our security architecture is described and important implementation issues are discussed, while in Section 5 the conformance of our architecture with the OGSA is addressed; in Section 6 we outline the testbed used for our experimentation and describe the different scenarios, measurement methodology and metrics used throughout our study. Finally in Section 7 comparative numerical results and relative discussions are presented, while Section 8 concludes the paper.

2. Related work

Generally, in security systems, the overall security problem when a client interacts with a server can be divided into three distinct functionalities: Authentication, Secure Message Exchange and Authorization. All these are closely related to the User Identity uniquely defined as: (1) username in password protected systems (2) the combination of the Distinguished Name along with the Issuer and Serial Number within a X.509 Certificate in Public Key Infrastructure – PKI equipped systems or (3) Kerberos Principal in Kerberized environments. Note that in Grids, credentials can also be related to Virtual Organization (VO) membership; in Kerberos based systems similar groupings to VOs are referred to as Domains or Realms (e.g. MS Windows Domains).

In legacy Grids where Globus middleware is employed, GSI (Grid Security Infrastructure) is the de-facto Authentication...
mechanism [20]. It is based on PKI Certification Authority issuing X.509 certificates. In order to support delegation the PKI has been extended by inserting the use of short-lived X.509 Proxy Certificates (RFC3820). The Trusted Third Party (TPP) is the Certification Authority (CA) that signs certificates of the user and the servers. In our architecture to the contrary we adopted the Kerberos authentication system in order to authenticate the users and support single-sign-on. Thus the users authenticate to the Kerberos system using their X.509 or proxy certificate with which they are identified in the Grid world (outside the Kerberos domain), and after their authentication they get a ticket from the Kerberos system with which they can gain access to various resources for the whole ticket duration without the need for re-authentication. As far as secure message exchange is concerned, it can be achieved with either the https protocol, i.e. http over Secure Socket Layer–Transport Layer Security SSL/TLS [21] or Web Services Security – WSS at the Application Layer [22]. In GSI secure message exchange is currently achieved mostly through SSL.

In our security architecture secure exchange of the SOAP messages is achieved through the use of WSS. WSS has an edge over SSL/TLS as an application level open specification. It provides confidentiality, integrity and non-repudiation at the message level by: (1) incorporating widely used authentication mechanisms into the SOAP message header and (2) performing XML Encryption or/and XML Signature to the required SOAP elements. Performance of the cryptographic algorithms used with WSS and the impact of the SOAP processing on these are presented in [23,24]. As shown in [25] however, there is a performance gain when SSL is used in comparison with WSS, partially due to the lack of overheads such as encoding keys and signatures written in ASCII. On the other hand, SSL presents limitations in restricted environments, e.g. with SSL a proxy server cannot provide re-routing functionality, as it may not read message headers. This is feasible with WSS at the application layer. Similarly, with SSL a message fails to activate services in a multi-hop path as it is encrypted on end-to-end transport level basis. This feature however is supported by SOAP and WSS. Furthermore, an important feature that also motivated the use of WSS in our implementation, is that it provides options on applying cryptography to different parts of a SOAP message (header and body). This allows the design of flexible authentication schemes, where various parts of the SOAP message can be encrypted by various intermediate services and decrypted at the destination. Such policies cannot be implemented by enforcing encryption at the transport layer i.e. sending the message via SSL/TLS. In other words WSS provides end to end security, whereas transport layer protocols provide point to point security, thus breaking the functionality of WS.

Finally, concerning the authorization, it is not part of GSI and it can be performed independently at the level of Computing Elements (clusters of Grid resources) using either Access Control Management techniques [26] or the Community Authorization Service - CAS [27] or the Virtual Organization Membership Service - VOMS [28]. Users of Grids can be grouped in Virtual Organizations (VOs) as defined in [29] i.e. interest groups requiring access to the same group of resources and services. VOs can be managed via a VO Membership Service (VOMS) server or in the case of CAS, the CAS server. This is accomplished by providing users with an attribute proxy certificate exposing their roles and the capabilities within his/her Virtual Organization (VO). Following these rules, we realized the authorization at the server side where the server checks against local policy rules if the user sending the message request is authorized to access the resource.

3. The web service security specification-profiles

As mentioned before, a primary objective for introducing the proposed security architecture is to improve the security performance and QoS of the service message processing by minimizing the overhead that security imposes on the overall system behavior. Since among our objectives is to design an OGSA [12] compliant architecture, that utilizes Web Services as Infrastructure layer, we followed the Web Service Security Specification (version 1.1), which enables applications to perform secure SOAP message exchanges with Kerberos credentials.

The core WSS specification defines an abstract security model to protect SOAP messages in terms of confidentiality and integrity, and authorize them using signatures and security tokens. Message protection is achieved by encryption of the body, or parts of the SOAP message, whereas integrity and message origin are verified by signatures which can sign either the header, or the body or any combination of them. The specification defines how the various security tokens are included in the message, but does not elaborate on how they are acquired by participating parties. These are implementation details and can vary among different security mechanisms and systems. In our case, we used the PKINIT to acquire the Kerberos credentials from the X509, as explained in detail in Section 4. All security related information targeted to a specific recipient is included in the WSS Header block, attached within the SOAP header.

Besides the core specification, there are additional WSS specifications on how to use both X.509 Certificates and Kerberos tokens. These are the WSS X.509 Token Profile and WSS Kerberos Token Profile respectively. They present extensions of the core specification to meet specific requirements of each security mechanism, i.e. they describe how to sign and/or encrypt SOAP messages using X.509 Certificates or Kerberos tickets respectively. In our architecture, we adopted the Kerberos Token Profile and we tested its performance against the X.509 Token Profile.

In the following we summarize both standards that we implemented and used in our experiments.

3.1. WS security X.509 token profile

This profile describes the use of a X.509 Certificate for WS Security. The standard introduces 3 types of BinarySecurityTokens containing the X.509 certificate. As an alternative, it specifies Token References (e.g. hash values) to the above Binary Security Token types in signature or encryption elements that comply with this profile.

3.2. WS security Kerberos token profile

The Kerberos Token Profile specification was first introduced in the version 1.1 of the WSS Specification. It basically specifies how to sign and encrypt a SOAP message by using a Kerberos ticket. Specifically, when exchanging messages to a service the Kerberos ticket should be included in the initial message; however, for performance considerations, the subsequent exchanged messages should include a token reference, referred as KeyIdIdentifier. The value of the reference should be the encoded (Base64) hash value of the Kerberos Ticket.

When the Kerberos ticket is referenced as an encryption key, the encryption algorithm must be a symmetric (secret key) one. The key used for signature and/or encryption is either: (a) a single session key issued by the Kerberos Server when a client wishes to contact a service or (b) a sub-key constructed from this session key. In our realization we adopted option (a). This session key is included inside the Kerberos ticket and is valid until the expiration of the ticket. The life-time of a ticket can be set by the Kerberos administrator; it can typically be up to 10 h, time that is comparable to the life span of an X.509 Proxy Certificate.
4. Architecture: Components and services

In this section, first the overall basic security architecture and corresponding principles that were followed are presented, and then the critical components and services of the architecture along with some implementation details are described.

4.1. Basic architecture and principles

The proposed Security Architecture has been designed to be easily deployed in a multi-domain environment where the user can authenticate to a Kerberos system once (single sign-on) and acquire credentials in order to call various Kerberos-protected Web Services securely by applying the WSS Kerberos Token Profile. Authorization is achieved by employing Access Rules at the Service Provider side and are enforced at the resources involved. The goal of the proposed security architecture is to provide performance improvement over legacy Grid security solutions (GSI), while maintaining interoperability with them. To achieve this, X.509 Certificates or Proxy Certificates - RFC 3820 are used for the initial authentication to the Kerberos Key Distribution Center (KDC) and then Symmetric keys are distributed to the involved parties (clients and Web Service Providers) for their subsequent communication. Due to the fact that Kerberos protocol is used after the initial logon, symmetric cryptography is utilized for all the cryptographic functions (encryption, signatures). It should be noted that authentication and authorization are performed on a per message basis and the security state (client–server session defined by a ticket) is maintained in both client and server sides per Client – Web Service pair. Thus clients do not need to re-present their credentials (issuing a new ticket) for as long as their Kerberos ticket is valid.

A high-level view of the security architecture is depicted in Fig. 1 and consists of the following components:

- **Authentication system**: provides the Kerberos authentication and key management functionality following the Kerberos Protocol.
- **KrbClient**: provides the API to the Client, hides the security complexity and manages user’s credentials.
- **Access Control Manager (ACM)**: protects the Web Service (e.g. IE in our case) by authenticating and authorizing incoming requests.
- **Policy Repository**: acts as a global repository where all the Local Access Rules are pulled. In an alternative operation mode it pushes the rules to the resources (IEs) thus, operating as a Policy Decision Point (PDP).

The high level description of the operation of the security system consists of the following basic steps. These steps are also highlighted in Fig. 1. Steps 3, 4 and 6 are optional, however if used they may improve the robustness of the whole operation.

1. The KrbClient when initialized, authenticates the user using his X.509 or proxy certificate to the Kerberos Authentication service. The Authentication service returns to the user a special ticket called Ticket Granting Ticket (TGT). For the subsequent calls there is no need to call the authentication service for the duration of the TGT. The duration of the TGT is chosen to be equal or less to the duration of the proxy certificate and in all cases its lifetime can be renewed in order to be valid throughout the whole operation cycle.

2. Following, the KrbClient, transparently to the Client, requests a ticket for the IE from the Ticket Granting Service (TGS). As a proof of Authentication, the KrbClient presents the TGT. If valid, a ticket for the IE is returned. As long as this TGT remains valid the client can request further tickets for other services without having to re-enter his credentials (single sign-on). A short expiration time of the TGT protects the system from old TGTs and Service Tickets that can be exploited by malicious users.

3. This is an optional step. The KrbClient can inquire the Policy Repository to discover which IEs or other web services he is authorized to invoke. The Policy Repository, as mentioned before, collects all access rules for all Web Services allowing for request and update of these rules.
4. If required, the KrbClient can delegate the client’s certificate to the delegation service. The delegated credentials can be used by the IE to access other Grid resources like Storage Elements and Computing Elements on behalf of the Client.

5. Having acquired the service ticket for the IE, the KrbClient communicates with a Web Service securely via WSS, sending a SOAP message with the acquired ticket to the Web Service or IE. A service ticket includes the client ID, client network address, validity period and user/server session key, encrypted by the service’s secret key. Thus only the target WS is able to decrypt the ticket. The secure communication between the server and the client is performed by the use of the session key, contained inside the ticket. The Client has the option for performance reasons to sign or/and encrypt the SOAP body. Encryption of the message using symmetric cryptography provides both message confidentiality and message integrity. We also note that signature means that the message is encrypted and then a signature of the encrypted message is attached at the SOAP message header.

6. Step 6 is triggered when a change to the local access rules is done. These rules are pushed to the Policy Repository. This is the default behavior of the Policy Repository. Depending on the need of each use case scenario the opposite direction has also been implementing, allowing the IE to pull their access rules from the policy repository.

4.1. KrbClient

The KrbClient is implemented as a Java API that performs all the security handling on behalf of the Client application in order to authenticate to the Kerberos system, acquire Kerberos credentials and request tickets for calling various WSs. Initially the client possesses an X.509 or an RFC 3820 proxy certificate with which he is identified in the grid community. VOMS proxy certificates are also valid provided that they follow the IETF RFC 3820. Using that certificate, the KrbClient API authenticates to the Kerberos Distribution Center - KDC via SSL and receives the Kerberos credentials (TGT). Kerberos uses for this initial login the PKINIT - Public Key Cryptography for Initial Authentication to Kerberos. Nevertheless the architecture does not exclude the use of other methods, such as username-password options. The Authentication Server - AS, running as a service inside the KDC, handles the authentication procedure. Once authenticated, a Ticket Granting Ticket - TGT is returned to the client.

For the Kerberos implementation we used the Heimdal Kerberos implementation [17] due to the fact that it supports the PKINIT. The KrbClient calls the Heimdal Kerberos kinit function that performs the initial login.

A pseudo code for the client implementation follows:

```java
Client implementation:
{
    Authenticate to Kerberos KDC();
    Request service tickets ();
    while (service_tickets are valid) {
        call services();
    }
}
```

The KrbClient after having been authenticated and acquired service tickets sends SOAP message requests to various Web Services by conforming to the WSS Kerberos Token Profile. This means that it adds a security header with the required Kerberos tokens as described in the above Profile. We implemented this functionality by extending the Apache WSS4J library in order to support the Kerberos Tokens because only X.509 tokens were supported by the WSS4J till the time of writing of this paper.

4.1.2. ACM

On the server side, to securely protect an Instrument Element or a Web Service in general, we have developed the Access Control Manager-ACM, implemented as an Apache Axis 1.3 Handler. Handlers can be deployed in front of a service and perform processing to the SOAP message.

The ACM, upon reception of a client request, checks the message and either verifies the signature if the message is signed, or decrypts the message if it is encrypted. After the successful authentication of the request, the ACM’s authorization component validates the message and approves or drops the request, based on the local rules. If the administrator of the WS changes the access rules, then automatically the new rules will be pushed to the Policy Repository for an updated view. The state diagram of the operation of the ACM is presented in Fig. 2.

The ACM has the ability to perform message authentication following the WSS Kerberos Token Profile. We have extended the WSS4J library to support the Kerberos Token Profile and added security state mechanisms to make the functionality robust. These state mechanisms hold the state of the session between the client and the server message exchange in order to facilitate and make the message interaction faster. For example in the initial message the client sends a whole binary security token to authenticate to the server, while in the subsequent interactions it sends only its hash given that the server has kept the binary security token from the first call.

4.1.3. Policy repository

The Policy Repository has the role of collecting authorization information or alternative pushing authorization information to resources. In the second operation mode, it acts as a Policy Enforcing Point (PDP) allowing the central management of authorization policies. Otherwise, the authorization policies are decided locally. The Policy Repository has been developed in Java, stores the rules in a mysql Database Management System (DBMS) and exposes its functionality as a web service. From the security perspective, we have protected it with an ACM that authenticates and authorizes the requests.

4.2. Identities and authorization

In this section we focus on the Authorization mechanisms and the Identity management of our security architecture. User authorization is tightly coupled with the policy of each site and the identity of a user. As stated earlier, the user ID can be the combination of a Distinguished Name (DN), the Issuer and the Serial Number in an X.509 certificate or the Principal in a Kerberized environment.

Access in our architecture is controlled by the Access Control Manager (ACM) handler. The ACM checks the user credentials against a local access list created by the local service provider administrator. In order to minimize the access rules, users are divided into groups and the access rules are referred to these groups instead of individual users. All access rules throughout our security system are uploaded to the Policy Repository. This global view helps to trace problems regarding authorization in the distributed environment.

In our architecture that we are dealing with Kerberos domains we identify a user or a service within a Realm (security domain) that he belongs to via a Kerberos Principal: principal_name/instance@REALM (following Kerberos V5 terminology). The principal_name refers to the username and the instance refers to the group that the user belongs to. A Realm can be viewed as analogous to a Virtual Organization (VO) in Grid terms. The use of Realms permits the existence of the same principals in different domains.
Within our security architecture a user can belong to various groups (instances). These groups are defined when the user identity is created at the Kerberos KDC. Groups are used for authorizing the user to the resources. If we map these groups to VOMS groups then authentication is achieved according to the VOMS roles.

The authorization to invoke an operation offered by a specific Web Service must be granted by its resource owner by default or centrally if the PDP operation mode of the Policy Repository is utilized. This authorization is given at the group level, and is not provided explicitly to each individual user. This approach is adopted in order to limit the number of access rules that exist within each ACM and therefore improve the efficiency of the authorization process. Note however, that access to resources and services must be explicitly requested from a resource owner – administrator of a Web Service. Using the Policy Enforcing Point (PEP) and Policy Decision Point (PDP) definitions, ACM authorization component can be considered as both PEP and PDP.

In our case access rules are defined as follows:

\[
\text{username group VO WS URL operation operationAttribute}
\]

where \(\text{username, group, VO}\) is the Kerberos Instance, WS URL is the endpoint of the WS or IE in our case, operation is the called operation and \(\text{operationAttribute}\) define the attributes that the operation accepts.

Fig. 2. ACM state diagram.
A rule allows a user belonging to a specific Instance (group) and REALM (VO) to call a Web Service. The default access policy is to deny access.

4.3. Mapping mechanisms

As we have analyzed our security architecture is applied to one or more security domains where Kerberos is used for message authentication. Initially the user has an X.509 certificate with which he is identified in the non-kerberized domains, and uses this X.509 to authenticate to the Kerberos system and acquire Kerberos credentials to be identified in the kerberized-domain. This translation of the X.509 to Kerberos credentials is achieved through the PKINIT, which is the main mapping mechanism that allows authenticating with a X.509 Certificate or Proxy Certificate. As a result the user is assigned a Kerberos identity as described above. However, as Kerberos is administered in realms that are closely related to VOs, there is a need to map not only an X.509 identity to a Kerberos one, but also VOMS proxy certificates. A VOMS proxy certificate is a super-set of an X.509 in which VO attributes are encoded. Thus, to permit the transparent and automatic mapping of an X.509 Certificate to Kerberos identity we first extract the user DN from the X.509 or VOMS Proxy Certificate. Then, this DN is hashed (via the MD5 algorithm), encoded (Base64) and used as a 224 bit Kerberos principal_name. This is necessary since the DN string format is not valid for a binary Kerberos principal_name, since it contains slashes ‘/’, spaces ‘ ’ etc.

As mentioned in the previous subsection (Identities and Authorization), the attributes that the VOMS proxy certificate defines can be set to agree with the access rules on each Web Service, thus providing authorization according to the VOMS roles. In this way it is possible to provide full interoperability between GSI-VOMS security and Kerberos.

4.4. Credential’s delegation support

The delegation procedure that has been described by Welch et al. in [30] is an important functionality in Grids as it allows a user to delegate a subset of his privileges to another entity. In a Grid environment that is used for the control of Instruments, delegation support is important as it allows IEs and WSs existing in our security domain to call other services and utilize grid resources on behalf of the user; an example is an IE to call a gridftp service in order to store acquired instrument data. The delegation is performed using the gLite Delegation Service.

In the following a brief explanation of the delegation functionality and how it is implemented by ACM is provided. Let us suppose that a service, e.g. Service_A, needs to call another service, e.g. Service_B, on behalf of a user. The user has to delegate his credentials to Service_A. This is accomplished by the user creating a proxy certificate signed by him and forwarding it to a Proxy Server. In order for Service_A to acquire this proxy and call Service_B it has to know the user DN. This DN is usually transferred from the client to Service_A via https (http over SSL) in GSI Grids. In our Kerberos domain though where no SSL is used, we need to add this DN in an extra security element inside the WS Security Header of a SOAP message heading from the user towards Service_A. At the same time, the user accesses Service_A, via our ACM handler, using his hashed DN as his Kerberos Principal. At the Service_A side, the ACM Handler performs an identical hashing operation on the DN piggybacked within the SOAP message and cross-checks with the Kerberos Principal of the user. If the user ID (DN) is compromised the delegation is cancelled.

Fig. 3 depicts our architecture in layers. The GSI Layer through proxy delegation allows the IE to call services outside our Security defined Domain. Starting from the bottom the different layers are the following:

The HTTP is the transfer protocol responsible for the transfer of the SOAP messages. The WS Security Kerberos Token profile defines how the Kerberos credentials are transferred inside the SOAP messages. Following are the Kerberos and the required mapping needed to authenticate the SOAP message at the IE. The GSI layer is the de facto security layer used in grids while the Delegation layer is used for the delegation of certificates to users/services to act on their behalf.

5. Conformity to the OGSA

OGSA defines the components that the Grid Systems should comprise in order to follow a Service Oriented Architecture (SOA) with web services as the main infrastructure. It does not define the low level mechanisms (e.g. Kerberos, PKI etc) that each component will use but provides only the high level view of the system framework pieces that make it interoperable. The
basic theory about this approach was set by Foster in [31]. The current version of the OGSA Architecture [12] defines a set of seven core capabilities that are required to support Grid Systems and applications: Infrastructure Services, Execution Management Services, Data Services, Resource Management Services, Security Services, Self-Management Services and Information Services. Each one addresses a specific domain of the Grid Infrastructure.

Concerning the Security Services [32], specific functional capabilities are defined, including Authentication, Identity Mapping, Authorization, Credential Conversion, Audit and Secure Logging. The architecture specifies an interoperable environment among different security domains that use different security principles, e.g. X509 certificates and Kerberos tokens and describes the services that allow this functionality. These functional capabilities are working automatically and transparently to the service requestor and service provider applications. It should be noted that the OGSA architecture does not mandate that the whole architecture is adopted, but a subset is sufficient to characterize a Grid Architecture as OGSA compliant.

Fig. 4 presents the components of the Grid Security model. We have highlighted in yellow the specific parts that our security architecture has implemented. Our schema is based on Kerberos credentials and incorporates exclusively symmetric cryptography, unlike X.509 based security. Conceptually, the entire infrastructure belongs to a separate security domain from the rest of the Grid. More specifically, our architecture provides:

- Credential and Identity Translation (Single Logon) through the Kerberos authentication system and the use of PKINIT to initially authenticate the users with their certificate and provide them with service tickets so that they can have access to various resources without having to re-login (Sections 4.1 and 4.1.1).
- Mapping Rules that allow the mapping of GSI certificates and RFC 3820 proxy Certificates to Kerberos Identities (Section 4.3).
- Access Control Enforcement via the Local Policies and the Policy Repository that discloses only the required view of the access rules for a specific user and is fine-grained per specific command and parameter of an instrument (sections 4.1.3 and 4.2).
- Service End-point Policy through the Local Access Rules (Section 4.1.2).
- Authorization Policy that allows only users of specific groups to access the instruments operations (Section 4.2).
- User and Key Management via the Kerberos Server KDC (sections 4.1 and 4.1.1).
- Binding Security by implementing the WS Security Kerberos Token Profile for transferring credentials and providing message security for the SOAP messages exchanged between the client and the service.

6. Testbed, scenarios and methodology

In order to assess our proposed security architecture and implementation, we developed a testbed and performed experiments to compare our Kerberos Token Profile Security implementation against the X.509 Token Profile Implementation. In this section, we first provide a brief description of the testbed configuration and the different scenarios that we used in our study, and then describe the measurement methodology and metrics used throughout our experiments.

6.1. Testbed description

The testbed consists of 1 server and 16 clients located in our Laboratory Local Area Network (LAN). The server is a Linux Debian 4.0 AMD64 (64 bit OS) running on an AMD64 X2 4.400 (2200 MHz), 2 GB of RAM, 2 × 120 GB hard disks in raid 1 configuration. The operating system is based on a Symmetric Multi-Processing (SMP) kernel. The client software runs on 16 Windows XP and Linux mixed environment.

The client implementation was written in Java 1.5. The clients have the ability to send messages with a specified rate and payload and encrypt the body of the SOAP message using AES128 and/or add a digital signature of the body. The server runs on a Java 1.5 64 bit Edition Java Virtual Machine (JVM); for the Web Services implementation we used Jakarta Tomcat 5.28 and Apache Axis 1.3. The security code of the server was written as an Axis Handler. Handlers can be deployed in front of a service and perform processing to the SOAP message. For the Web Services Security (WSS) X.509 Token Profile we used the default AxisHandler coming within WSS4J (Web Service Security for Java). For the WSS Kerberos Token Profile we used our ACM handler. Both implementations require a Java Cryptography Extension implementation in order to provide the cryptographic functionality. To satisfy this requirement the BouncyCastle [33] JCE provider was utilized.

6.2. Scenarios

Our main goal is to measure the performance of the security processing at the server using message authentication, under
high CPU load (over 80%). Performance was assessed by measuring the average message throughput (SOAP messages/s), the effective throughput and security specific processing times. These metrics are defined in detail in Section 6.4.

As it has been shown in previous work [34], when comparing encryption of WSS X.509 and WSS Kerberos token profiles both exhibit similar behavior in terms of same message throughput under low CPU load. Differences start to emerge when the CPU is saturated, allowing for better performance of the Kerberos implementation in processing more SOAP requests. In use cases of Grids controlling and instrumenting numerous resources, the bottleneck is usually the server performance as it is overwhelmed with large numbers of short messages requiring low latency. The more messages per second the server processes the better average per message response time we obtain, thus achieving smaller latency times. For testing purposes in our scenarios, a simple Web Service was chosen (echoing client strings), in order to avoid other processing overheads that might distort our measurements, as our emphasis is placed on the comparative performance of WSS profiles.

Two different scenarios, referred to as SC1 and SC2 in the following, have been considered throughout our study. Both involve SOAP message authentication and message integrity via signing (the message was not altered). However, SC1 also guarantees message confidentiality via encryption of the SOAP message body. Both scenarios involve measurements of message throughput and security related processing times at the security handler. The reason for testing both these scenarios is that although they are based on the same cryptographic principles and is expected to exhibit similar qualitative behavior, it is quite important to measure/quantify their relative performance differences, as such results could be useful to determine which method to use in various applications (message authentication with message integrity or message authentication with encryption), in order to satisfy their performance and security requirements.

It is noted that we have not included the server’s policy component in our measurements, since the time for the retrieval of the policies from the database in each application scenario would remain the same for a specific user, and therefore the authorization as a component would influence the performance at a constant factor. Consequently, throughout our measurements we emphasize on depicting the difference on the performance of secure message exchanges, since these correspond to the heavy part of the security processing, by performing cryptographic functions.

6.3. Methodology

The methodology that we followed in both scenarios involves 16 clients sending to a single server fixed length SOAP messages, at their maximum achievable rate. Each client sends a total of 10,000 messages (overall 160,000 messages in our system). All clients start sending messages at the same time. The measurements are repeated with varying message payload sizes, namely 60, 400, 800, 1600 and 3200 bytes each time. The size of the messages in our remote control operation is generally small control messages (< 1000 bytes) and therefore no fragmentation over the network is realized in these cases. However, in our measurements we have included some indicative messages of larger size (e.g. 1600, 3200 bytes) which undergo fragmentation in order to observe whether and how the delay is affected. It should be noted here that these values refer to message payload size before encryption. To assess the server performance, we measured the number of SOAP messages and the corresponding processing times for each handler for the duration of the experiments. Only the receiving direction to the server has been protected, since it is the direction involving a WS invocation by clients. The specific details of methodology adopted in the different scenarios for each mechanism are the following:

6.3.1. X.509 Certificates

In SC1 scenario, each client first encrypts and then digitally signs the body of the SOAP message with a random symmetric key using the AES128 algorithm. The random symmetric key is encrypted using RSA15 with the public key of the referenced server’s certificate. The encrypted body is attached to the WS security header along with a reference to the issuer and to the serial number of the server’s X.509 certificate. Then the signature is also attached to the security header along with a reference to the issuer and serial number of the client’s certificate.

On the server side, under SC1 scenario, the X.509 WS Security handler retrieves the service’s private key and the client’s public key from the PKI keystore, where all the certificates are stored. Then, using the private key it decrypts the attached RSA15 encoded symmetric key. Subsequently, it decrypts the body with the attached symmetric key, verifies the signature with the client’s public key, thus authenticating the message, and finally passes the decrypted message stripped from the security header to the actual service.

With reference to the SC2 measurements, the client digitally signs the body of the SOAP message, creates the WS security header and attaches the signature with a reference to the issuer and the serial number of the client’s X.509 certificate. When the SOAP message reaches the server, the handler retrieves from the PKI keystore the client’s public key by looking at the referenced key and verifies with it the attached signature.

6.3.2. Kerberos tokens

The client logins to the AS of the Kerberos system and requests from the TGS a ticket for a service that he is authorized to access. Then, it attaches this Kerberos ticket to the security header. Subsequently it uses this Kerberos ticket for applying encryption and signatures to the SOAP messages for the two different scenarios (SC1 and SC2).

More specifically, in the SC1 case, the client encrypts and signs the SOAP body using the session key derived from the Kerberos Token. In the subsequent messages, the client encrypts the body using the session key and attaches to the security header a reference to the Kerberos ticket instead of the whole one. This reference is the SHA1 hash value of the Kerberos ticket.

At the server side, when the handler accepts the message, it first verifies the signature, extracts the service ticket, derives the session key and finally stores it in a hash table. This key defines a session at WS Security layer for the subsequent client–server transactions. After that, the handler decrypts the body with the session key and forwards the decrypted SOAP message to the Web Service. In subsequent messages it retrieves the session key from the hash table by checking the reference to the Kerberos token sent by the client.

With reference to SC2 scenario, the client attaches a digital signature of the SOAP message. In the Kerberos Token Profile this signature (referred to as Hash Message Authentication Code HMAC) is actually the hash value of the body of the SOAP message encrypted with a symmetric algorithm (AES128 in our case).

When the handler at the server receives the first SOAP message, it creates the session at the WS Security layer like in SC1, performs message authentication by validating the attached signature with the session key and forwards the message to the actual service. In subsequent messages, the handler retrieves the session key like in SC1 from the hash table by checking the reference to the Kerberos token.

6.4. Metrics

The scenarios and the methodology described above aim at studying the behavior and operation of both mechanisms under
heavy load with different SOAP body sizes. The processing times that are recorded by each security handler are the following:

- Total Handler Processing: Total time of the WS Security processing (at the handler). This time is actually the addition of all the times shown below.
- SOAP Processing: Time that the handler needs to get a SOAP message and de-serialize it.
- Security Request Processing: Time to parse all security tokens (Kerberos ticket, X.509), derive encryption keys from token references, verify signatures and decrypt a message (if applicable).
- Request to Axis: Time required to reconstruct the message by removing the security elements and (if applicable) by substituting an encrypted message (ciphertext) with a decrypted one (cleartext).
- Verify header, cert, timestamp: Time for the final verification of the header, certificate (cert) and timestamp, if present.

All the above times concern the processing performed at the security handler, after which the processing of the actual service along with the preparation of the reply may take place. Due to the fact that only the above times vary in each case (X.509 and Kerberos) and all other relevant times remain the same for both mechanisms, the other relevant times are not included in our measurements as they do not affect the security performance of our handler.

Apart from the above times, in order to obtain a better understanding of how each handler performs, we measure the Average SOAP Message Throughput (messages/s) served by the Web Service during the whole duration of the experiment. The Average SOAP Message Throughput is of particular significance in delay sensitive Grids, as sessions usually require a large number of small messages in order to minimize latency and delay variation (i.e. delay jitter). Based on this we also calculate the Effective Throughput as follows:

\[
\text{Effective Throughput} = \left( \frac{\text{Average Throughput}}{\text{SOAP Body Size}} \right) 
\]

where SOAP Body Size refers to the message size before encryption.

---

Table 1

<table>
<thead>
<tr>
<th>SOAP Body size</th>
<th>60</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>3200</th>
</tr>
</thead>
<tbody>
<tr>
<td>X.509 (messages/s)</td>
<td>226</td>
<td>222</td>
<td>218</td>
<td>209</td>
<td>195</td>
</tr>
<tr>
<td>Kerberos (messages/s)</td>
<td>342</td>
<td>332</td>
<td>324</td>
<td>298</td>
<td>273</td>
</tr>
<tr>
<td>Gain%</td>
<td>51</td>
<td>49</td>
<td>48</td>
<td>43</td>
<td>40</td>
</tr>
</tbody>
</table>

The Effective Throughput, as defined above, expresses the real data sent from the client to the server, excluding the various overheads. The Effective Throughput is lower than the actual throughput, since it does not include the SOAP Header, Security Header, HTTP header, TCP header and IP Header. However, these headers can be considered to have almost the same size for both Kerberos and X.509 Token Profile mechanisms, and therefore they can be excluded from the comparison of the two security handlers.

7. Measurements—Results

In this section, we present comparative numerical results between the WS Security X.509 Certificate Token profile and the WS Security Kerberos Token profile, for both scenarios described in 5.2.

7.1. Scenario SC1—encryption + signature

First, we examined the service performance utilizing Kerberos and X.509 handlers to perform message confidentiality and integrity via encryption and signature.

Fig. 5 presents the variation of message throughput using both encryption and signature as a function of the body size. We observe that there is a relatively small degradation in the performance of both solutions, as the size of the SOAP body increases. The following Table 1 presents specific numerical values of the results depicted in Fig. 5, as well as the relative average throughput gain of Kerberos over X.509 based solutions. The corresponding gain is defined as: Gain% = \((X.509 - \text{Kerberos})/X.509\) * 100%.
As observed in Fig. 5 and Table 1, the Kerberos solution offers significant gain in the average message throughput especially for small values of the message body size. Specifically, the Kerberos solution achieves 51% gain over the X.509 at 60 bytes body size. This gain is due to the faster derivation of keys in Kerberos implementation, since symmetric cryptography is used, while asymmetric cryptography was adopted to derive keys in the X.509 handler. This difference is reduced with large body size (e.g. 40% for body sizes of 3200 bytes). This is due to the fact that both X.509 and Kerberos handlers employ symmetric decryption and as SOAP messages grow larger, body decryption becomes a more significant factor than key derivation.

Similar observations and conclusions can be drawn, as expected, by examining the behavior of the Effective Throughput as a function of the SOAP body size (Fig. 6). Based on the results of Figs. 5 and 6, we also notice that the whole process of both handlers is mainly limited by message throughput, rather than effective throughput.

In Figs. 7 and 8 we depict the average values of the various security processing times described in Section 6.4, for the Kerberos and for the X509 Certificate handlers respectively. We observe that the total security processing time for the X.509 Certificate handler case is almost double than the one for the Kerberos case. This difference in performance is mainly attributed to the Security Request Processing time illustrated.
by the red color. The corresponding improvements observed in the Kerberos implementation are due to the fact that the X.509 Token mechanism uses symmetric keys retrieved with asymmetric cryptography, while the Kerberos Token mechanism is solely based on symmetric keys.

We have also calculated the standard deviation ($\sigma$) in both measurement data and they were approximately 25 ms for the Kerberos handler and 38 ms for the X.509 handler for all body sizes. The standard deviation indirectly reflects the jitter (delay variation) which is of particular interest in delay sensitive Grids. Again from the standard deviation we notice the advantage of Kerberos over the X.509 handler.

7.2. Scenario SC2- digital signatures only for authentication & message integrity

Figs. 9 and 10 illustrate the performance gain of our Kerberos Token Profile implementation over the X.509 Token Profile under scenario SC2. The advantage of Kerberos using signatures for message authentication and integrity is evident. For instance, our proposed solution achieves to reach an average message throughput of 473 messages/s, for SOAP messages of 60 bytes long. Of course this comes at the expense of message confidentiality provided by message encryption in SC1. However, in cases where confidentiality is not a critical issue, and we rather require high performance along with message integrity both Kerberos
and X.509 provide the required functionality, with Kerberos outperforming X.509 solutions.

Figs. 11 and 12 present the corresponding security processing times of Kerberos and X.509 handlers using Authentication and Message Integrity via Signatures. These are in accordance to the performance results we presented for SC1 scenario. However, they exhibit lower processing times as they do not need to encrypt the SOAP body as in SC1.

8. Conclusions

In this paper, a client–server model for a grid security architecture that follows the OGSA guidelines and provides enhanced near real time services in Grid applications by adopting symmetric cryptography during the actual operation, has been introduced and designed. This architecture is targeted to Grids with increased requirements in terms of throughput and response times (e.g. instrumentation Grids), because it delivers better interactive functionality.

The Security Architecture was designed and prototyped, implementing the WSS Kerberos Token Profile as it exhibits better performance than the X.509 Certificate Token Profile in SOAP message authentication and integrity. The architecture incorporates standard GSI features such as single-sign-on authentication, delegation of credentials, message integrity (via digital signatures), and confidentiality (via encryption) and can add authorization of messages.
on user requests. For compatibility with GSI legacy schemes, initial authentication of users is based on their X.509 or VOMS Proxy Certificates. Furthermore, our Architecture provides a full mapping of GSI credentials to Kerberos ones.

The main advantage of our Architecture is due to the adopted Kerberos Token profile that handles security more efficiently than the X509 one at the SOAP level, thus providing higher message throughput. The corresponding measurements demonstrated that the implementation of the Kerberos Token profile consistently exhibits up to 50% message throughput improvement over the X.509 Token profile, under full CPU load on the server. This difference is mainly attributed to the symmetric cryptography that the Kerberos implementation employs throughout a message exchange session, including the key derivation phase. It is this superior performance in handling high rates of short messages that renders the proposed architecture attractive for applications with real time constraints such equipment, instrumentation and control Grids, which are typically characterized by a high volume of short messages, unlike bulk data use cases, typical in legacy Grids.

References


Athanasios Moralis holds an Engineering Degree from the National Technical University of Athens - NTUA (2000). He is currently a full time researcher at the Network Management & Optimal Design Laboratory (NETMODE), School of Electrical & Computer Engineering, NTUA. Since his graduation, he was involved in several National & European projects focused on Security and Grid Computing. Other research interests include, Intrusion Detection Systems and P2P computing.

Vassiliki Pouli is a Research assistant and a PhD student at the Computer Science Division of the Electrical & Computer Engineering Department of NTUA (NETMODE laboratory). She received the diploma in Electrical and Computer Engineering at the National Technical University of Athens (NTUA) in 2005. Since 2005, she has been a teaching assistant in the course “Network Management and Intelligent Networks of the Department of Electrical and Computer Engineering, NTUA. Her main research interests are in networking, security, service level agreements and negotiation of agents.

Symeon Papavassiliou is currently, with the Faculty of Electrical and Computer Engineering Department, National Technical University of Athens (NTUA). Before joining NTUA he was an Associate Professor at the New Jersey Institute of Technology (NJIT), USA, while from 1995 till 1999 he was a Senior Technical Staff Member at AT&T Laboratories in New Jersey. He was awarded the Best Paper Award in INFOCOM’94 and the National Science Foundation (NSF) Career Award in 2003. He has an established record of publications in the field of computer and communication networks, with more than one hundred technical journal and conference published papers.

Vasilis Maglaris is currently a Professor in the Electrical and Computer Engineering Department, National Technical University of Athens. From 1979 to 1981 he was with the Network Analysis Corporation, New York, while in 1981 he joined the Department of Electrical Engineering at Polytechnic University, where he was an Associate Professor. He served as the Chairman of the Board of the Greek Research & Technology Network - GRNET, the National Research & Education Network (NREN) of Greece, from September 1998 until June 2004. Since October 2004 he was elected as the Chairman of the National Research & Education Networks Policy Committee.