A Security Architecture using Symmetric Cryptography and Kerberos-based approach for Performance Improvement in Grids[1]

A. Moralis, V. Pouli, M. Grammatikou, S. Papavassiliou, V. Maglaris

Network Management & Optimal Design Lab (NETMODE)
National Technical University of Athens (NTUA)
9 Iroon Polytechniou str. Zografou, 15780, Athens, Greece
{amoral, vpouli, mary, papavass, maglaris}@netmode.ntua.gr

Computing Grids that are traditionally used for batch computations are extending their application on different areas. In GRIDCC [4], we extend the Grid paradigm to support distributed instrumentation that belongs to different organizations. The instrumentation exposes its functionality as Web Services. Quality of Service (QoS) plays an important role to applications of that kind, thus QoS in the design and functionality is required by all the components of the GRIDCC. In order to minimize the impact on server performance of security processing, we have designed and implemented a security architecture that is based on certificates for the initial login, uses Kerberos to distribute symmetric keys and provides message level security implementing the OASIS Kerberos Token Profile [14]. In this paper we present the implementation details and some measurement we have performed that validate our expectations from the architecture.

Keywords: Dynamic Management on Grids, Certificates, Authentication Authorization Infrastructure, , Web Services Security, Web Services Security Kerberos Token Profile

[1] This work was partially supported by the EC GridCC Project (IST 511382). The authors performed this work within IASA (Institute of Acceleration Systems & Applications), a joint Institute of the University of Athens & NTUA

1. Introduction

Distributed Kerberized Access Architecture (DKAA) presented in [19] aims at extending Grid middleware to provide improved performance, and also secure Authentication and Authorization Infrastructure (AAI) for distributed systems. A remote, distributed over the net, interactive and multi-user system is subject to threats encountered in resource-sharing systems. These include unauthenticated and unauthorized access to resources, session hijacking, replay attacks, Denial of Service (DoS) attacks, etc.

DKAAs' main goal is to provide a balance between a mechanism that offers access control, confidentiality and non-repudiation by authentication, authorization and encryption - thereby eliminating most of the threats mentioned above - for low latency usage of large distributed Grid resources. Therefore, the security architecture should specify a light-weight scalable Authentication & Authorization Infrastructure (AAI), along with efficient procedures for secure message exchanges.

DKAA was designed as the AAI and secure data exchange of the GridCC project [4]. GridCC aims at developing a Grid framework that enables the remote control of instruments with real time or near real time constraints. Their successful operation often requires rapid interaction with computing – storage resources and a large number of instruments. The real time and interactive nature of instrument monitoring and control requires the provisioning of acceptable quality of service among various Grid components. Furthermore, since the shared instruments may belong to different organizations, the implementation of a high performance security scheme that allows dynamic management of Virtual Organizations is of paramount importance.

The paper is organized as follows. Section 2 provides a brief overview of existing popular AAI architectures that can be applied in a collaborative Grid environment, while section 3 describes the proposed Distributed Kerberized Access Architecture. Section 4 defines the authorization mechanisms. In section 5 the implementation of the DKAA in GridCC is presented, and finally Section 6 concludes the paper.

2. AAI Infrastructures for Grids

Grid Security Infrastructure [7] (GSI) is the de-facto Authentication protocol used in Grid environments [8]. It is based on Public Key Cryptography [9] (PKI), using X.509 certificates. An important extension of the PKI is the use of proxy certificates. Through proxy certificates, single sign-on

and delegation is achieved. The authorization is not part of GSI and it can be performed independently at the level of Computing Elements (cluster of Grid resources, e.g. instrument controller in GridCC) using CAS [13]. The Trusted Third Party is the Certification Authority (CA) that signs the user and hosts certificates. Secure message exchange can be achieved through encryption. GSI is easily deployed and managed, as each organization manages its users' certificates. Therefore trust among the different organizations is only required. Alternatively, authorization in Grids can be achieved by the employment of Virtual Organization Membership Service (VOMS) [11]. VOMS provides to users an attribute proxy certificate that exposes the roles and the capabilities of each user within his/her VO [12]. Then the user uses this certificate to access the resources. The service acknowledges the user's authorization by checking the attributes in the VOMS proxy certificate.

When designing the DKAA, we thought of maximizing the performance of the Security Infrastructure. In this respect, GSI, as all PKI based solutions if used exclusively, presents an important drawback due to the heavy overhead of public key cryptography that it employs, especially in applications that require low latency. If an entity that has specific QoS requirements accepts numerous requests, the substantial higher CPU load of asymmetric (public) key cryptography can reduce the overall performance in a great extent. Nevertheless, GSI, as a PKI based system, is suitable for a Grid environment, with a multitude of users and resources.

3. Distributed Kerberized Access Architecture (DKAA) Overview

DKAA is designed to provide secure communication to Web Services and offers the same functionality that GSI presents, with the extension of handling authorization and authentication in the security layer and not in the application layer. This allows DKAA to be relatively easily deployed in front of any Web-Service. In Figure 1, the general DKAA architecture, along with its basic elements and components are presented.

The security architectural choices stem from the need to cover distributed environments that give emphasis on providing secure real time (or near real time) access to resources (e.g. instruments) for monitoring and control purposes, with acceptable Quality of Service (QoS) guarantees.

Kerberos [2] is based on the Needham-Schroeder Protocol [15]. Kerberos offers authentication and single sign-on features using symmetric key cryptography. The authentication process is performed by the Authentica-

tion Server (AS) in conjunction with the Ticket Granting Server (TGS). Both of these servers run as services within the Key Distribution Server (KDS). In addition, there are mechanisms that map X.509 certificates to the Kerberos Authentication System, in order to maintain interoperability with certificate based systems. PKINIT [6] is an RFC that allows the use of certificates for authentication to the Kerberos system. Furthermore, the GSS API that is the API used to access a Kerberos system is defined in detail in [3].

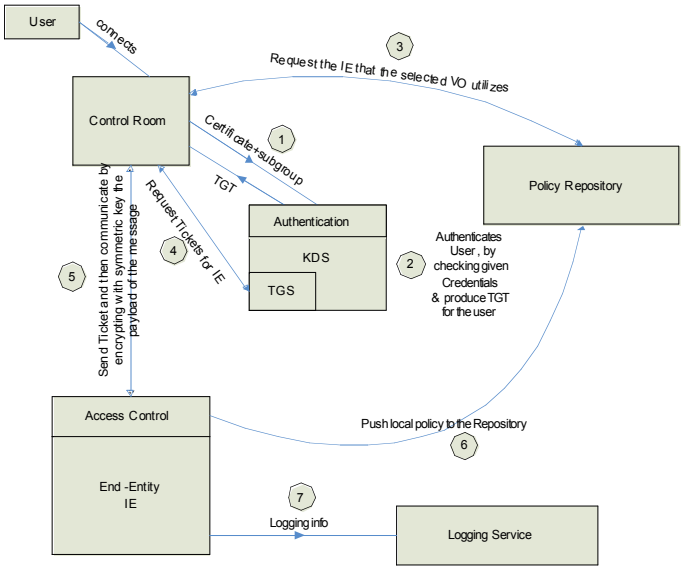

Figure 1: Architecture Overview

Once authenticated, a Ticket Granting Ticket (TGT) is returned to the user. When a user wants to access a service located at an end-entity, a DKAA aware client requests a ticket from the TGS transparently to the user. The ticket, when presented to the end-entity performs mutual authentication. Ticket based authentication is based on symmetric key cryptography and thus the authentication setup imposes lower overhead to the control clusters in contrast to Public Key Cryptography.

In the case of Public Key Cryptography, a PKI asymmetric scheme establishes a secure session after a computationally expensive handshake procedure (SSL - Secure Socket Layer) imposing a penalty factor that varies form 3.5 to 9, as shown in [5]. Although resuming a session via caching

of the results of a previous SSL handshake helps reducing the problem, it does not alleviate it completely, as the initial setup introduces performance penalties, especially under heavy load. In some GridCC Use Cases, thousands of instruments might be involved, each requiring secure message exchanges. The problem is more intense if sessions are terminated in a proxy (e.g. Computing Element or Instrument Element) instead of the terminal resources (e.g. "worker nodes" or instruments) whereby a large number of sessions is concentrated in a single point.

Using Kerberos, as a centralized authentication system, renders the authentication procedures relatively easy via the Key Distribution Server (KDS) acting as a Third Trusted Party. Management of users and roles is significantly simpler than a distributed scheme that relies on several PKIs with revocation lists difficult to maintain. In order to cope with the fact that KDS may be a single point of failure, failover KDS systems should be used, so that the availability of the distributed system is guaranteed. Similarly, security of the KDS itself is critical, thus it must be well protected and monitored. Based on Kerberos' single sign-on provisioning capability, a user is authenticated once (gives his/her credentials) and uses multiple resources and services over a period of time (e.g. one hour). This function is implemented by the limited lifetime of the Kerberos tickets, issued by a Ticket Granting Server. After expiration of a ticket, re-authentication is needed. Depending on the application, life-times can vary from minutes to even a day. This duration should be determined, based on time statistics of a control session.

Both Kerberos and GSI are used for authentication purposes and session encryption key negotiations. User authorization is tightly coupled with the policy of each site. It is handled by the Service Provider via a separate system, DKAAs' Access Control Manager (ACM). The ACM checks the users' credentials against a local access list created by the site owner. In order to minimize the access rules, users are divided into subgroups and the access rules are referred to subgroups instead of users. All the access rules of all the service providers are uploaded to the Policy Repository. This global view helps to trace problems regarding the authorization in the distributed environment.

DKAA provides a client API that hides the complexity of the Kerberos protocols and manages the secure message exchange, offering different levels of security (requested by the application). If secure message exchange is required, the client and the service provider can use the session key (incorporated within the ticket) to encrypt the message or part of the message. By default DKAA appends a digital signature to the message, thus authenticating and preventing alterations by other parties. The level of

encryption can play a significant role to achieve the performance goal that each system based on DKAA poses. Based on the above criteria and objectives, we outline the guidelines of a baseline security architecture as follows:

1. An end-user is authenticated by using an X.509 Certificate issued by a trusted Certification Authority to the KDS. A proxy certificate should be fine. This provides single sign-on functionality. An authenticated user is authenticated once per session and accesses many resources.
2. The core system consisting of clients and service providers (or end-entities) employs a ticket-based authentication mechanism and symmetric cryptography. Tickets are acquired via centralized Key Distribution Servers.
3. Authorization is performed at the service level employing local access rules.

Following this approach, in the remaining of this section, we present and describe the various basic components of the DKAA architecture.

Authentication Server (AS): AS handles the authentication procedure. The client sends a request to AS, along with the user's credentials. These could be a trusted certificate or username – password. Other types of credentials (like one-time passwords) could be used as well. The authentication procedure is encrypted, using SSL. The AS checks the user's credentials and if they are found correct, a TGT is returned. This TGT among others contains user-specific information, referred to as principal. The principal contains the user name and the subgroup that he/she belongs to (called instance in Kerberos terminology). The user credentials are also stored in the AS.

Ticket Granting Server (TGS): TGS grants tickets to a requesting service. The client must present the TGT received from the AS. After checking, the TGS grants a ticket for the required service. The ticket contains the user principal. This function is performed transparently to the user.

Policy Repository: It collects local policies for all services. The policy of each service is set by the site owner that the service resides. It is used to locate the existence of specific subgroups within the various services. This is especially important for a single sign-on procedure.

Access Control Manager: It is a software layer similar to an application layer firewall. It filters all incoming connections to the service in two stages:

- Authentication of the received message: The first time a user accesses the end-entity, within a control session, he/she sends the ticket he/she owns to access the end-entity. The ACM decrypts it with the end-entity

key and stores locally the session key along with the user principal (username/subgroup) and the expiration time of the session key. This is done only once for the duration of the session key. After that, authorization of the messages is performed with the decryption of the content with the session key.

- Authorization to the requested service: The end-entity checks the local access rules set by the site owner, on the requested service; in conjunction with the user's subgroup it accepts or denies access to the service. This is done per request.

Logging Service: It holds the audit logs. Every transaction with the service providers is recorded, along with its status. The status could be "success" or "failure". A transaction may include, but not limited to, the User/subgroup, the command, the status and the reason.

4. Users and Authorization

DKAA is designed to allow users to authenticate to resources that permit access and control on distributed resources (e.g. instruments). Therefore the overall infrastructure has to support communication and interactions between two types of entities: *Users* and *Resources* (e.g. instruments). Each resource provides some functions to the users, which in turn are translated into services. The users control resources by using the corresponding services.

The identity of a user or a service is uniquely identified in the simplest form with the use of a *principal/instance@REALM* (following Kerberos V5 terminology). The use of a REALM permits the existence of the same principals in different domains. A REALM can be viewed as analogous to a domain or a Virtual Organization (VO) in Grid terms. The principal value refers to the username and the instance value refers to the subgroup that the user has log in.

These identities are valid within the DKAA. Outside, the user is identified by his/her X.509 certificate. PKINIT allows authenticating the user with his/her X.509 Certificate to the DKAA and as a result the user gains the Kerberos identity as described above.

Within the DKAA a user can belong to various subgroups (instances). These subgroups are defined when the user identity is created at the Kerberos KDS. Subgroups are used in DKAA for authorizing the user to the resources.

The creation of a subgroup and the specification of its users and services, do not automatically provide authorization to the resources. Access to

the resources and services must be explicitly requested from the resource owners (administrators where the service is provided).

The authorization to invoke a service offered by a specific end-entity must be granted by its resource owner. This authorization is given at the subgroup 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 end-entity and therefore improve the efficiency of the authorization process. However, if granularity down to the user level is required by a use case, a separate subgroup can be created for each user. In this way, the access control can be extended to support access to the user level.

The format of an authorization access rule could be as follows:

Service	Serv Provider ID	Subgroup

The above description represents a general form/representation. In case that, services are provided in the form of a Web Service [10], the rule can be refined to match with the Web Service architecture as follows:

Service: portType, Operation

Serv-Provider ID: Service Endpoint URL or Service Location

Subgroup: Kerberos Instance or Group

Therefore, according to the above terms an access policy rule may take the following form:

Operation	PortType Service	Endpoint Url	Kerberos Instance

The default access policy is default deny. This means that if specific access rules are not explicitly defined, the service is denied to any subgroup requesting it. Therefore, effectiveness and simplicity is maintained in enforcing end-entity access rules.

5. DKAA Implementation and Results

DKAA is being developed as the Authentication & Authorization Infrastructure (AAI) of the GRIDCC project [4]. To this end, a new resource has been defined, namely the Instrument Element (IE), similar to the Computing and Storage Elements (CEs, SEs) of the batch Grid. The IE acts as the middleware that enables the remote control of instruments over the network as services. With the addition of supporting services, like the Workflow service and the Agreement service, a user of the system will be able to automatically control instruments and define the Quality of Service

characteristics of his/her interactions. To offer simplicity to the users, a Virtual Control Room (VCR) has also been defined that acts as the User Interface of all the services in the GRIDCC collaborative environment.

The IE is exposed as a Web Service and the software is written in java using Apache AXIS [16]. The VCR is also developed in Java. Thus, security should be able to integrate easily with these components. For all these we chose Java as our implementation language.

Consequently, we have chosen to use the OASIS Web Services Security [14]. We have developed an implementation of the WS Security Kerberos Token Profile [20], by extending the Apache WSS4J java library [18]. WSS4J supports the Web Services Security and specifically implements the X.509 Certificate and Username Token profile but it does not implement the Kerberos Token Profile yet.

In Figure 1 the general architecture was presented. To meet these designed goals we have implemented the following components:

- KrbClient: is a java object that provides the client security API needed to login to the Kerberos Server (KDC), acquire service tickets and apply signatures or encrypt the messages. It sends the messages as SOAP messages. Specifically, it inserts into the SOAP header the WS Security Header, implementing the Kerberos Token Profile, attaches XML signatures and encrypts the body using XML Encryption. The goal of the KrbClient is to facilitate the development of DKAA clients by hiding the complexity of the security processing from the application developer. This involves hiding all the Kerberos and WS Security details. In addition it holds all the security state such as user tickets and session keys with the various services. It uses the Java GSS API, Heimdal kinit [17] and WSS4J for supporting all the security processing.
- ACMHandler: performs the access control and all the security service logic of the IE and it is deployed as an AXIS handler. It preprocesses the SOAP messages destined to the service and it specifically checks the security header, authenticates the messages, verifies the possible signatures and decrypts the body, if it is encrypted. Handlers are deployed at the request flow of a web service and are transparent to the service. Finally the ACMHandler authorizes the message by checking against the locally stored Access Rules. If the security processing is successful it forwards the processed message to the actual service.
- Policy Repository: is an LDAP that stores all the local policy rules, allowing the monitoring of the global authorization scheme distributed among the IE.

In the following we demonstrate the operation of the basic elements of the DKAA architecture in the GRIDCC security Use Case, by identifying and describing the various interactions and exchanges involved during the authentication and authorization phase. In a typical GRIDCC scenario there are at least two different organizations that have IEs. Let us consider the scenario illustrated in Figure 2. In this scenario we assume the existence of two organizations: Organization1 and Organization2. All other supporting GRIDCC entities have been omitted for simplicity in the presentation. These two organizations are independent in every aspect, including policy and users. Organization1 has two Instrument Elements, IE1 and IE2, while Organization2 has only one, IE3. The two organizations have agreed to share part of their resources to users belonging to Virtual Organization 1 (VO1). Specifically, Organization1 provides IE1 and Organization2 provides IE3. VO1 also sets up its own Key Distribution Server (KDS1) that has all the credentials (GSI certificates) of the users belonging to VO1.

Passwords are created for IE1 and IE3. These passwords are only known to their corresponding IEs and the KDC. Supposing that user@VO1 tries to send control commands to IE1 and IE3. The steps involved in this exchange are:

1. The user logs in the Control Room. The user submits his/her credentials (GSI certificate) and the subgroup that he/she wants to use. Transparent to the user, the KrbClient of the client API logs in to the KDS1 using the user's certificate. In case that the credentials are valid, the log in, is successful and a TGT is returned.
2. Let us assume that the user sends commandA and commandB to IE1 and IE3 respectively, through the Control Room interface, choosing the security scheme (either the body of the SOAP message is encrypted or a signature is attached). Then, transparently to the user the following actions take place:
   - The KrbClient requests tickets for IE1 and IE3 from the TGS by providing the TGT that was acquired in step 1.
   - The KrbClient sends the service request (CommandA and CommandB) as a SOAP message, attaching a security header and the tickets as Binary Security Tokens, to the IE1 and IE3. The tickets are contained in structures called AP_REQs which hold the ticket along with an authenticator and are placed inside the Binary Security Tokens, as mentioned before.
   - The ACMHandlers of IE1 and IE3 process the header of the incoming SOAP message and extract the ticket. If it is accepted the user is

authenticated and the Session Key (SK1 for IE1 and SK3 for IE3) is stored locally at the ACM of IE1 and IE3, along with the user and his subgroup. The Session Key (SK) is contained inside the ticket and is valid until its lifetime expires. After storing the SK, the ACMHandler using the SK decrypts the SOAP body if encrypted or validates the signature. When using symmetric cryptography the signatures are Hashed Message Authentication Codes (HMAC). Finally, before forwarding the message to the actual service, the ACMHandler authorizes the user that calls the service by checking against the locally stored access rules.

- The service processes the request and sends back the reply.

3. The Control Room displays the result of the user's commands (success or failure).

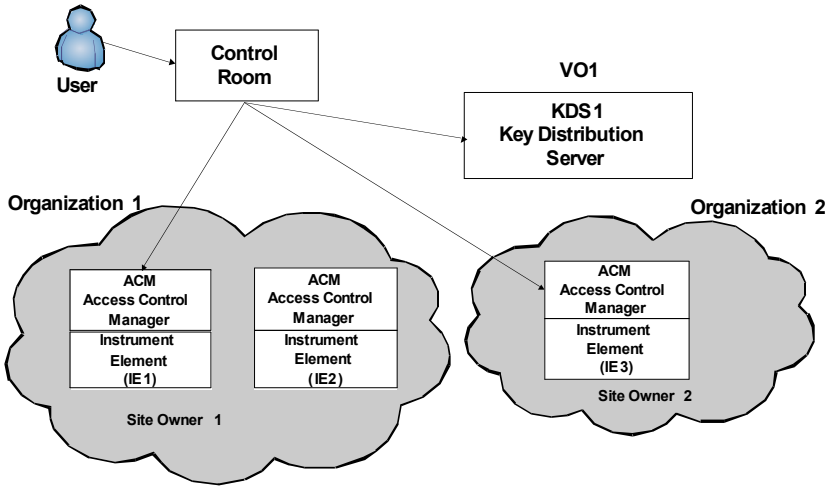

Figure 2: A simple Use Case

4. In the subsequent messages of the user to IE1 or IE3 the above procedure is performed using the Session Key for encryption and signature. It is not needed that the ticket be attached (in the form of an AP_REQ) but for performance reasons a reference to the ticket is used. This reference is the hash value of the AP_REQ. In the scenario described above, it is assumed that the access rules have been set locally by the administrator of each Organization. This step is performed prior to the creation of the subgroup. If there are any changes in local policies, they are pushed to the Policy Repository. Access rules that allow access for the subgroup of users are set only in IE1 and IE3. In the case that the user sends a

command at IE2, although he/she can be authenticated, the lack of an access rule for his subgroup at IE2, results in denying access to it.

In the framework of this work we measured the performance of the ACMHandler by flooding it with requests and compared it with the default handler that WSS4J provides. The default handler supports the X.509 Certificate Token Profile whereas ACMHandler implements the WS Security Kerberos Token Profile. We send messages to both handlers with encrypted body, using AES128 symmetric algorithm. The X.509 handler used RSA (Public Key Cryptography) to encrypt an attached symmetric key that was used for decryption. The CPU utilization of the service process was over 95%. For testing purposes, we have chosen an echo web service, which accepts a string and replies sending that string. This service is of low complexity, as the main objective is to picture the performance of the WS security by minimizing the load from other factors, like the ones from the service. The payload of the request, that was encrypted, was 200 bytes. In the Table 1 we present the results:

WS Security Profile	Average Packets/sec
Kerberos Token	104.86
X.509 Token	82

Table 1: Performance of WS Kerberos Token Profile compared with X.509 Certificate token Profile

The performance gain of our Kerberos implementation is 27.9%. This performance difference is mainly attributed to the way that each method acquires the key that is used for decrypting the message. Our implementation uses the session key that is inside the Kerberos ticket. This is extracted using symmetric cryptography. In contrast, the implementation of the X.509 Token, derives the symmetric key by decrypting the attached encrypted with RSA (Public Key Cryptography) key in the SOAP message. After acquiring the symmetric key, both implementations decrypt the body that is encrypted with AES128.

Additionally, by changing the packet size as shown in Figure 3 we can observe the greater packet throughput achieved by our implementations in contrast with the X.509 implementation for different body sizes. Both performance of the ACMHandler and the X.509 handler degrades slightly with the performance difference sustained at the same level. We have not tested body sizes greater than 800 bytes, as we wanted to avoid the fragmentation of the of the SOAP message in more than one IP packets. The size of messages destined to control apparatuses should always be kept minimum so that latency is avoided.

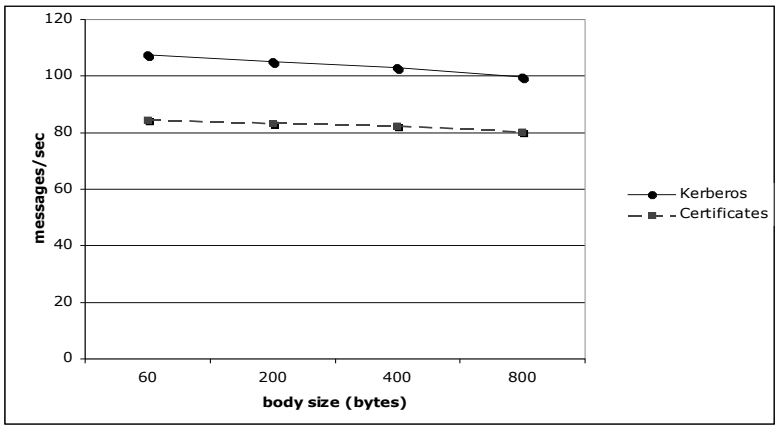

Figure 3 Average packet rate for different body sizes.

6. Conclusions

In this paper we presented a Distributed Security Infrastructure, namely DKAA that aims to minimize the overhead of cryptography and improve the QoS of the overall system. It achieves that by using exclusively symmetric cryptography for the message exchanges, when interacting with a large number of Grid resources (e.g. distributed monitoring and control of time critical instruments). The architecture implementation has been adapted with the Web Services paradigm. DKAA provides authentication, integrity, confidentiality and authorization at the SOAP-message layer.

Symmetric Key Cryptography approach has lower computational complexity than Public Key Cryptography. Therefore, it allows the overall service to offer lower response time and better performance compared to alternative Public Key Cryptography approaches. This performance improvement reflects to our measurements and shows clear advantage in overloaded servers when DKAA is used.

References

1. The European Policy Management Authority for Grid Authentication in e-Science, http://www.eugridpma.org/

2. IETF RFC 1510- The Kerberos Network Authentication Service (V5)
3. IETF RFC 1508 - Generic Security Service Application Program Interface
4. GRIDCC Project web site - www.gridcc.org
5. C. Coarfa, P. Druschel and D.S. Wallach, "Performance Analysis of TLS Web Servers", 9th Network and Systems Security Symposium, pp. 553--558, 2002
6. IETF RFC 4556 - "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)"
7. Grid Security Infrastructure - http://wwwunix. globus.org/toolkit/docs/3.2/security.html
8. Global Grid Forum - http://www.ggf.org/
9. IETF RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile.
10. W3C Web Services Activity http://www.w3.org/2002/ws/
11. R. Alfieri et al, "VOMS, an Authorization System for Virtual Organizations", Presented at the 1st European Across Grids Conf., Santiago de Compostela, Spain, Feb. 14, 2003
12. R. Alfieri, R Cecchini, V. Ciaschini, F. Spataro, L. Dell'Agnello, A. Frohner, K. Lorentey, "From gridmap-file to VOMS: managing authorization in a Grid environment", Future Generation Computer Systems. Vol. 21, no. 4, pp. 549-558. Apr. 2005
13. L. Pearlman, V. Welch, I. Foster, K. Kesselman and S. Tuecke, "A Community Authorization Service for Group Collaboration", IEEE Workshop on Policies for Distributed Systems and Networks, 2002
14. Oasis WS Security Standards, http://www.oasis-open.org/specs/index.php#wssv1.1
15. Roger Needham and Michael Schroeder, "Using encryption for authentication in large networks of computers", Communications of the ACM, 21(12), December 1978
16. Apache AXIS, http://ws.apache.org/axis/
17. Heimdal Kerberos Server, http://www.pdc.kth.se/heimdal/
18. Apache WSS4J, http://ws.apache.org/wss4j
19. A. Moralis, A. Lenis, M. Grammatikou, S. Papavassiliou & V. Maglaris, "A Distributed Kerberized Access Architecture for Real Time Grids", 4th International Workshop on Security In Information Systems WOSIS, 2006
20. WS Security Kerberos Token Profile, http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf