Using SPKI Certificates for Authorization in CORBA based Distributed Object-Oriented Systems

Tuomo Lampinen

Helsinki University of Technology
Department of Computer Science
FI-02015 TKK, Espoo, Finland
tuomo.lampinen@hut.fi

Key words: CORBA, SPKI, certificate, authorization, access control

Abstract: CORBA based middleware has been used for the last couple of years mainly for bringing the old legacy applications into the web age, but now this role has begun to change, as new applications are built on top of it. Together with this change, legacy based access control along with other security functionality has to be converted from the centralized mainframe world into the distributed Internet world. This change needs solutions which are originally designed for distributed environments. Among these solutions are SPKI authorization certificates defined by the IETF working group.

In this paper, we present a way of implementing authorization in CORBA based distributed applications with SPKI certificates. We discuss the potential advantages of this approach compared with traditional access control list based solutions and also describe an architecture which we have implemented in our project.

1. INTRODUCTION

The powerful communications infrastructure provided by the Internet has for the last few years made it possible to build globally distributed applications. The Internet has also set new requirements for application architectures demanding a different level of modularity, flexibility and scalability. Various middleware-based architectures have emerged and attempted to overcome these problems by providing the application developers with new architecture solutions. Middleware is used to provide a distributed run-time environment for reusable software components. These components typically offer well-defined and uniform interfaces to various back-end systems and databases thus hiding the heterogeneous hardware and software environments used in the legacy systems and making thin-client architectures possible. The platform independent Java programming language and component-based design have been the two driving forces behind this middleware revolution.

So far, collecting business logic into reusable components and wrapping legacy applications have been the main uses for middleware, but as the products and technologies mature, more and more of the critical business applications are moving into the middleware based architectures and losing their ties to old legacy systems. This means that the middleware based applications that have depended upon the legacy systems for their security have to implement equivalent security services as those provided by the legacy systems. However, simply reimplementing the old security services is not the best approach, because the Internet is decentralized unlike the old mainframe-based enterprise world. Securing the communications traffic in the Internet is often based on the Secure Sockets Layer (SSL), but few good alternatives for access control in Internet-like distributed environments have been proposed.

One possible alternative for distributed access control is the use of SPKI certificates, which in addition of having many other desirable properties also support anonymity and privacy, both of which have been jeopardized by many other developments regarding the Internet. The goal of this paper is to evaluate a SPKI certificate based access control architecture, which is suitable for large scale distributed applications. Our implementation is based on CORBA, which is an open industry standard for distributed computing infrastructure. The access control architecture also shares the key benefits of CORBA – it is language independent and interface centric thus offering a lot of freedom for actual implementation.

1.1 Authorization in distributed environment

Traditionally authorization and access control decisions are based on the use of access control lists (ACLs). Access control lists are usually implemented as database entries, in which each resource is followed by a list of subjects who are allowed to access the resource [5]. Access control lists are a well-suited solution for centralized client/server systems, but they do not scale well in distributed systems. The use of ACLs in distributed systems usually means either that we have a central storage, which easily forms a single weak point and bottleneck in a large system, or that we replicate the ACL database to multiple distributed servers, which usually leads to difficult update and synchronization issues, especially because the integrity of the ACL database is important and must thus be protected against unauthorized modifications. Access control lists also favour the use of globally unique names mapped to access rights, which makes it hard to support anonymity and privacy with ACL based approaches. Furthermore, access control lists lack the possibility to delegate authorizations to a third party, which is an important quality in distributed systems. For example, an information service in the web could sell access rights to the information either directly, or through retailers, who could delegate the access rights further to paying customers.

All the weaknesses of access control lists are also inherited into distributed applications, if their security, especially authorization and access control, is built on top of ACLs. This is why we have implemented an alternative architecture based on authorization certificates, which avoid most of the weaknesses related to ACLs. Authorization certificates are conceptually similar to capabilities in that they are used to map subjects to objects, but unlike capabilities certificates are digitally signed, which protects them against unauthorized changes.

The rest of this paper is organised as follows. Section 2 introduces CORBA and the concept of distributed object computing. Section 3 discusses the security features of CORBA. In Section 4 we introduce SPKI certificates and their advantages over more traditional approaches in distributed access control. Section 5 outlines our approach for implementing SPKI certificate based access control architecture in distributed CORBA environment. Section 6 gives a more complete description of our architecture implementation. Finally, Section 7 evaluates the implementation against the given criteria and discusses some ideas for further development. Section 8 presents our conclusions.

2. DISTRIBUTED OBJECT COMPUTING AND CORBA

Distributed object computing utilizes the paradigms of object oriented programming for implementing distributed applications. This supplies distributed applications with the advantages of object oriented programming such as polymorphism, inheritance and encapsulation in distributed environments.

2.1 Distributed object computing and middleware

Distributed object computing is based on the idea of separating the interface of an object from its implementation, as shown in Figure 1.

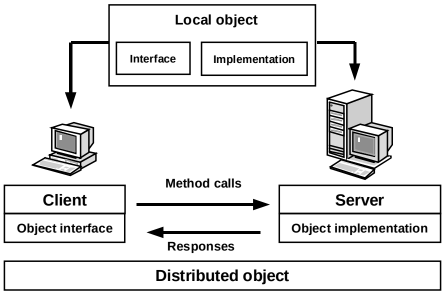

Figure 1. The concept of a distributed object

This seemingly simple concept has powerful implications. Separating the interface means that clients using the services provided by the interface no longer have to know where the actual implementation of the object is located or how the implementation is done. These properties are usually referred to as location transparency and implementation transparency, respectively. Because the implementation details no longer matter to the client, the actual implementation of the object may even be done in another programming language and run on a different hardware platform in another computer.

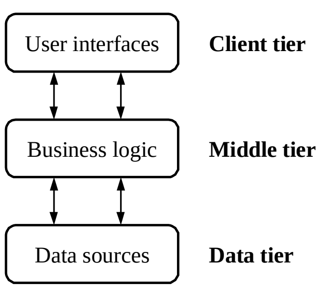

Figure 2. A typical multi-tier architecture

Distributed object systems are by nature multi-tiered. A typical division of tiers is presented in Figure 2. This division separates presentation, business and data management logic into three logical tiers, which serve as the basis of the software architecture. Business logic is typically implemented with reusable software components in the middle tier. These software components are run in middleware products, which offer the necessary services for the software components such as managing the lifecycle of the components, providing transactions, handling security related issues and offering administration services for the components. The client tier typically consists of the user interface and presentation logic. Data sources on the other hand can be databases, legacy applications or other external interfaces.

In the traditional client/server computing model the middle tier and data tier are usually located on a single physical machine and only the client tier is distributed. Contrary to this, in distributed object computing model all tiers can be distributed, which results in many preferable properties including:

- **Flexibility:** the logical tiers can be divided in many ways in heterogeneous hardware and software environments. The implementation of whole tiers can be changed without affecting the rest of the architecture as long as the interfaces remain the same.
- **Scalability:** separating the tiers offers better resource sharing and load balancing between multiple middleware servers along with thin-client options. Specialized back-end machines can also be used to provide highly efficient data sources.
- **Robustness:** data sources and middleware servers can be replicated making fail-over possible thus providing high availability systems.
- **Reuse:** separating business logic from user interfaces and data sources into independent software components offers many opportunities for easier reuse. Also many common features in applications such as security related services need only be implemented once thus reducing the risk of incorrect implementations.

2.2 CORBA

One of the most popular middleware architectures is Object Management Group's (OMG) Common Object Request Broker Architecture, i.e., CORBA [10]. CORBA is currently an industry standard for open distributed computing architecture. Some of the main strengths of CORBA are its language and platform independence, which make it well suited for heterogeneous environments such as the Internet. Compared with some other middleware technologies, CORBA is also quite mature a technology and has already proved its value in enterprise environments. CORBA has good support for distributed object computing paradigms like location and implementation transparency and provides a well defined set of basic object services [8] for objects such as object lifecycle handling, events, transactions and security. However, CORBA itself is only a specification and it is left up to vendors to transform the specification into working, interoperable products called Object Request Brokers (ORBs). An ORB is an object bus, which allows client objects to communicate with remote objects by invoking their methods. The communication between ORBs is based on the Internet Inter-ORB Protocol (IIOP), which enables interoperability between different ORB vendors

3. SECURITY AND ACCESS CONTROL IN CORBA

Distributed systems are by nature more vulnerable to security breaches than the more traditional systems, as there are more places where the system can be attacked. Compared to traditional client/server systems, security in distributed object systems is also more challenging, because distributed objects can play both client and server roles, so a simple division between trusted server components and untrusted client components no longer works.

3.1 The CORBA Security Service

Security in CORBA is handled by the Security Service, which belongs to the CORBA Object Services [8] defined by OMG. The Security Service defines a framework with functionality for authentication, authorization, encryption and auditing, thus meeting the basic requirements for protecting sensitive data and controlling user access in applications. The interfaces defined are generic enough to allow the use of many different underlying security technologies to be used in securing CORBA applications. The Security Service can be implemented to conform to one of the two levels of security:

- **Security Level 1** is meant for applications which are unaware of security and for those having limited requirements to enforce their own security in terms of access control and auditing.
- **Security Level 2** has all the features of Security Level 1 and also allows applications to control the security provided at object invocation and includes administration of security policies.

CORBA security is based on the following objectives [8]:

- Maintain confidentiality of data and system resources.
- Preserve data and system integrity.
- Maintain accountability.
- Assure data and system availability.
- Provide security across a heterogeneous system where different vendors may supply different ORBs.
- Provide purely object-oriented security interfaces.
- Use encapsulation to promote system integrity and to hide the complexity of security mechanisms under simple interfaces.
- Allow polymorphic implementations of objects based on different underlying mechanisms.
- Ensure that object invocations are protected as required by the security policy.
- Ensure that the required access control and auditing is performed on object invocation.

As these objectives show, the CORBA Security Service is intended to be very flexible. Especially, the CORBA Security Service Level 2 offers applications many options for customizing the security implementation. This customization is based on a security context object called *Current*, which represents the current execution context in a client/server interaction and can be used to obtain both the client and server credentials. This context is checked when a method invocation request enters the ORB environment and on exit of those requests on the object implementation side, as shown in Figure 3. This security context can also be extended for application specific security needs, for example, to provide customized audit logs or access control. Because the security context related to method invocations is implicit, changes to applications' business logic are minimal.

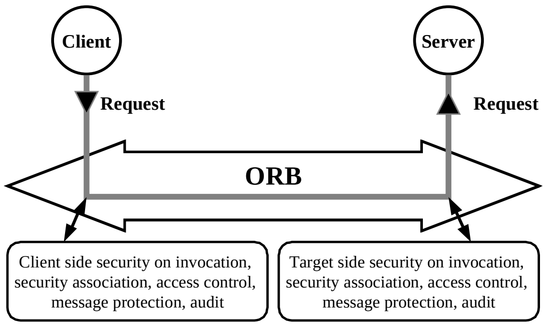

Figure 3. Method invocation related security options [8]

Access control in CORBA is checked by an *AccessDecision* object, which is the CORBA world version of the reference monitor concept [5]. The *AccessDecision* object provides a method called access_allowed, which determines whether the client's invocation is allowed. The access decision chain is shown in Figure 4. This access control can be performed by object or by method basis. Extending this interface is the basis for customized authorization and access control implementations as described in Sections 5 and 6.

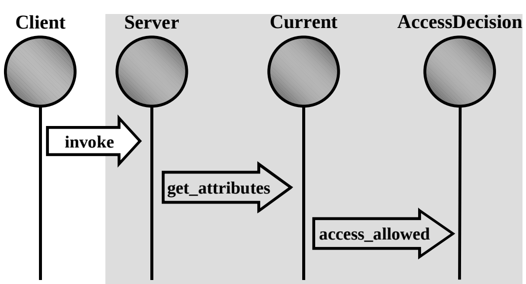

Figure 4. Access decision in invocations [16]

3.2 The usage of CORBA Security Service

Despite the rich set of features offered, many CORBA based distributed applications don't take advantage of the Security Service. The most common reasons for this are the following:

- Lack of proper Security Service implementation in ORB implementations – Many CORBA products do not provide a proper implementation of the CORBA Security Service Level 2.
- Only encryption of network traffic is used – Many CORBA products offer easy encryption of inter-ORB network traffic with IIOP over SSL [9]. This has the advantage that the developer usually doesn't have to make any changes into the application code, because SSL provides transparent encryption to end-users.
- Legacy application based security – Many CORBA based applications avoid authentication and authorization issues by simply tunnelling the user credentials to the legacy systems, which provide the necessary services for authorization and access control. This is usually the case in financial applications developed in banks and insurance companies where the security is largely implemented into the legacy systems.

However, new applications being developed are becoming more distributed and severing their ties to old centralized legacy systems. This means that we can no longer trust that all security issues are being handled by legacy systems in the visible future. Using SSL with IIOP takes care of the important issue of securing the inter-ORB traffic of remote object method invocations, but another major security issue, namely access control, is still left to be implemented. Simply porting access control implementations and policies from the centralized legacy systems into the distributed object world is often not the best approach, because many access control solutions that work well in centralized systems become cumbersome in distributed systems lacking the necessary scalability [11]. One of the well-researched alternatives for implementing access control in distributed environments is the use of digital certificates, which are signed statements about the properties of entities. Because of the flexibility offered by the CORBA Security Service regarding the implementation of the access control policy, digital certificates can be adopted to function as the basis of access control in CORBA.

4. SPKI CERTIFICATES FOR AUTHORIZATION

A public key infrastructure (PKI) is a system providing a mechanism for publishing public-key values bound to some other pieces of information such as a name or an authorization. To support the interoperability of applications, PKIs define certificate formats and semantics, as well as the process of verifying that a certificate is valid. [18]

4.1 SPKI certificates

Simple Public Key Infrastructure [1, 2, 3] is an Internet draft standard which defines public key certificates for authorization. SPKI is intended to provide mechanisms to support security in a wide range of Internet applications and to solve many of the problems regarding authorization in distributed environments. Some of the main ideas behind SPKI certificates are its emphasis on decentralization and its usage of public keys instead of names as principals [2]. SPKI certificates don't set any restrictions on the permissions that can be defined. Some examples about SPKI certificates are given in [3]. The SPKI certificate structure is shown in Figure 5.

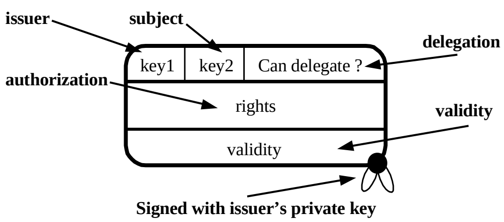

Figure 5. SPKI certificate structure

An SPKI certificate is a signed message which consists of five security relevant elements [1]: issuer, subject, delegation, authorization and validity. The certificate is signed by issuer's private key and grants the specified authorization to subject. The validity field describes the conditions under which the certificate can be considered valid. This validity field is usually given as a time range, but other on-line conditions can also be used. The delegation field offers boolean control for delegation defining whether the authorization can be delegated by the subject to a third party.

4.2 Advantages of SPKI certificates in distributed environments

SPKI certificates have some important characteristics which make them suitable for being the basis of authorization in distributed environments. These advantages include:

- Decentralization – Certificates can be issued freely by anyone, so issuing certificates is not restricted to a central authority. Many other standards, such as the X.509 standard [15] assume that there is a single certification authority (CA) hierarchy creating certificates thus limiting the trust model.
- Delegation of authorization – Access rights can be delegated, in which case a chain of certificates is formed. Delegation of access rights makes the system much more flexible. For example, parents having authorization to a certain bank account could delegate this authorization to their children, while at the same time limiting the maximum single withdrawal amount to a value specified in the authorization field.
- Flexible permissions – Authorizations and permissions can be freely defined and are not restricted to any predefined set. However, while this approach provides maximum flexibility, stardardization of common permissions is preferable in order to promote certificate interoperability. Some examples of possible certificate formats are given in [3].
- Validity – The issuer of the certificate can specify a time period or other on-line conditions under which the certificate is valid. This offers more fine-grained control for delegation along with easier access rights maintenance, and can be used to minimize potential damage in case a certificate is compromised.
- No name bindings – Because certificates are bound to keys instead of names, they can be used directly without first finding a public key corresponding to a given name. Binding certificates to keys also offers a better protection of privacy, since certificates can be used anonymously when a subject generates a temporary key for the issuer to be used in a certificate. These temporary identities can be used to promote privacy as discussed in [12].

These advantages combined make SPKI certificates a powerful alternative to access control lists (ACLs) discussed in Section 1. Unlike ACL databases, certificates can be published without any encryption. When a certificate is used in a request, the request is signed by the subject's private key in order to prove that the user is entitled to use the certificate. Because certificates are signed, they can't be modified or used by any third party even if they are completely public. This means that certificates can be distributed directly to end-users. In large-scale systems, where delegation may form long certificate chains, it is probably preferable to store part of the certificates to some distributed storage. However, because this database doesn't require heavy protection against modifications, existing technologies such as Domain Name Servers (DNSs) can be used for storage [17].

5. IMPLEMENTING AUTHORIZATION WITH SPKI CERTIFICATES IN CORBA

There are no doubt many possible ways of combining SPKI certificates with CORBA Security Service. Issues such as the granularity of access control and transparency to application developers can be tackled in many ways. While our approach tries to be as general as possible, other solutions may work better for different kinds of CORBA environments.

5.1 Environment

The model of our environment is based upon the situation shown in Figure 6. It consists of a server offering resources to retailers, retailers reselling the resources of the server, and clients, who wish to buy and access the services offered by the resources. This kind of environment sets authorization requirements, which can easily be solved by using SPKI certificates. Basically the server gives an authorization certificate to a retailer, which delegates the authorization provided by the server's certificate to clients against a fee.

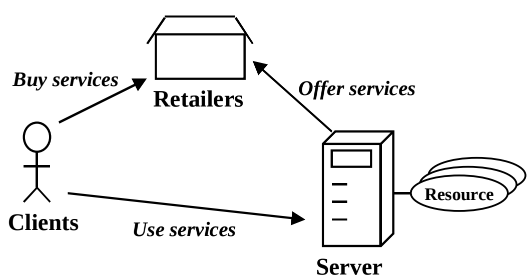

Figure 6. Typical use case for the architecture

This use case model is transformed into the CORBA environment shown in Figure 7, which is used in our implementation. The environment consists of five main components, which are resource server, reseller, client, resolver and resource factory. In a typical configuration there are multiple instances of clients and resellers.

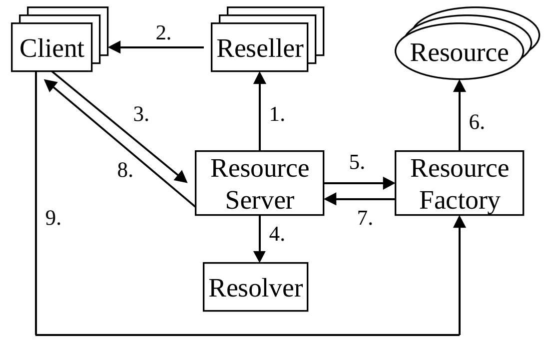

Figure 7. CORBA environment

Resellers are third parties, which sell the access to the resources to clients. This is done by delegating the certificate obtained from the resource server to the client after the client has provided its public key. Delivering the certificates to clients can be done, for example, over the Internet, in which the client could pay for the certificate with anonymous electronic cash.

Clients represent the end-users, who are interested in the services provided by the resources. After the client has got its certificate from a reseller, it stores the certificate locally and uses it in all subsequent communication with the resource server. This means establishing a connection with the resource server and requesting an object reference to a resource object.

Resource server is the server which handles the initial interaction with clients requesting the resources. The resource server works as a proxy for the resource factory, and is responsible for filtering out unauthorized requests. In case that the client's authorization for the requested resource is verified successfully, the resource server requests a new resource object instance from the resource factory and passes its reference to the client.

Resolver is responsible for the hard work of trying to create a valid certificate chain providing authorization for the requested resource. This means finding a chain in which the issuer of the first certificate is the resource server and the subject of the last certificate is the client requesting the resource. The resolver also performs checks to verify the validity field of each certificate and to analyze the relationships between the permission tags in the certificates. Typically only a small piece of this chain is provided by the client, so the resolver has to use some sort of a storage from which it fetches the certificates when trying to form the chain. Examples of this kind of storage would include DNS based solutions [17, 18] or X.500 directory [14] and LDAP [19] based solutions.

Resource factory is responsible for creating and managing instances of resource objects. The resource factory also accepts the requests for the resource instances from the clients and performs the desired online checks for each call. Physically the resource factory may run on the same machine as the resource server, but this is not mandatory.

Resource objects offer service interfaces for various computing resources. Some possible resources for this kind of systems would be digital content servers, data processing services on a large supercomputer or information services. In addition to implementing a common resource interface, each resource object also offers a resource specific service interface, which provides the necessary services for the clients.

5.2 Implementation choices

One of the issues to be decided in this kind of a system is the granularity of the access control. CORBA based applications typically take advantage of the factory pattern [6], in which one object called the factory or home is responsible for creating, finding and managing instances of one class of objects. One example would be a *BankAccountFactory* object, which would provide a method called *FindAccount*. *FindAccount* would take an account number as a parameter and initialize a new instance of the *BankAccount* class based on the data fetched from the host's database. Therefore a factory object seems a natural place for adding the access control. The permission tag in the client's certificate contains the resource name, i.e., the resource factory class, and the parameters for the resource creation, for example, *(tag (resource (name "BankAccountFactory") (account "12345678") (operation "ViewBalance") (operation "ViewTransactions")))*. In the environment described in Figure 7 the resource server is responsible for verifying the authorization, after which it passes the parameters in the permission tag to the given resource factory. Because factories are usually used to create nearly all server side objects, the granularity of access control can be freely chosen by determining which factories require an authorization in order to be used.

Perhaps the largest issue in the implementation of the architecture is to decide when and where to use the CORBA Security Service functionality. The starting point of the architecture is to use CORBA Security Service whenever possible, because this avoids redundant work and makes most of the security implementation transparent to the application developers by avoiding the need to pass security related information explicitly. This means using the security context, i.e., the *Current* object, to transparently pass the certificates from each client to the server. This has the advantage that each client can simply add all its available certificates for the application in question to the security context as a start up procedure. After this the client can use the services provided by the server without having to add additional parameters to each method call for passing authorization information. This approach is shown in Figure 8.

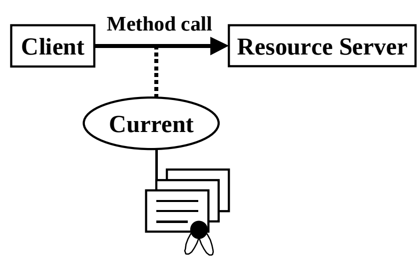

Figure 8. Passing certificates in the Current object

After the method call is received by the server, all security aware objects on the server side are responsible for obtaining the client's certificates from the security context and performing the necessary checks before granting access to the resources. This can also be done transparently, by using a customized access control policy for the factory objects. The access control policy implements the *AccessDecision* interface, and uses the resolver to verify that the certificates obtained from the security context of the method invocation are a part of a valid certificate chain. If the *AccessDecision* object can verify the authorization with the help of the resolver, it lets the method call pass to the resource server.

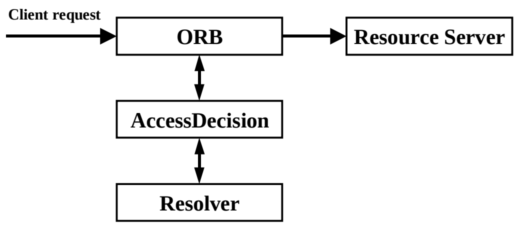

Figure 9. Checking authorization on the server

Another issue in the implementation is to decide how the client proves that it is the rightful owner of the public keys for which the certificates have been issued. Normally this would be done with a challenge-response approach, in which the client signs some piece of information provided by the resource server during the connection establishment. ISAKMP [4] could also be used for this purpose.

6. IMPLEMENTATION DESCRIPTION

Our implementation relies on standard CORBA features and is implemented in Java. Due to the limitations concerning the availability of ORBs supporting the CORBA Security Service Security Level 2 outside the US, we have implemented some parts of the CORBA Security Service specific functionality ourselves. The use of SPKI certificates and the resolver is based on existing implementations [7, 17, 18], so this section describes only the CORBA specific parts of the implementation. The implementation also assumes that the network communication between the components situated on separate physical machines is encrypted and protected against unauthorized modifications. The easiest way of performing this is to use IIOP over SSL, but other solutions such as IPSec [13] could also be used if they are supported by the environment.

6.1 Adding certificates to the Current object

During the initial start up procedure of the client application the client's SPKI certificates are added to the security context. For simplicity and because determining the actually required certificates for a particular method invocation is hard, all certificates are added to the security context at once. The certificates are added as security attributes to the client's credentials object, which is contained in the *SecurityLevel2::Current* object. For this the certificates have to be transformed into their encoded transmission format, in which they are represented as simple byte sequences. The client performs the whole operation by first creating its credentials with the *authenticate* method of the *SecurityLevel2::PrincipalAuthenticator* object. Because we are interested in authorization, we can setup the authentication to use anonymous SSL authentication. After this the client obtains its own credentials through the *own_credentials* attribute of the Current object. Finally the actual certificates are added to the credentials by calling the *set_privileges* method for it and passing the certificates as security attributes. When this is done the client can start using the services on the server and the certificates are always transparently passed along with the method call parameters.

6.2 Verifying the certificates

On the server side we have implemented an *AccessDecision* object, which takes care of passing the received certificates to the resolver for verification. The *AccessDecision* object is automatically called for each method call, when it is set as a new security policy for the resource server by calling the *set_policy_overrides* method for the resource server. This means that authorization checks are performed automatically before the method call reaches the resource server object.

The IDL interfaces for the access_allowed method and the resolver are shown in Figure 10.

```
// AccessDecision::access_allowed
boolean access_allowed(
  in SecurityLevel2::CredentialsList cred_list,
  in Object target,
  in CORBA::Identifier operation_name,
  in CORBA::Identifier target_interface_name
);

//  Resolver interface
interface Resolver {
  // Verify the certificate chain from the issuer to
  // the subject for the given resource. Return true
  // and the verified chain on success or false if
  // verification  fails.
  boolean verifyChain(
    in  PublicKey              issuer,
    in  string                 resourceName,
    in  SPKICertificateSequence subchain,
    out SPKICertificateSequence verifiedChain);
};
```

Figure 10. AccessDecision and Resolver interfaces

The *access_allowed* method gets the client's credentials, from which the certificates can be extracted by calling the *get_attributes* method. The *AccessDecision* object then calls the Resolver, passing as parameters the resource server's public key, the resource name corresponding to the object for which the operation is invoked and the extracted certificates. The job of the resolver is to find a certificate chain for the resource from the issuer to the subject. Typically the minimum length of this chain is at least two certificates, one from the resource server to the reseller, and one from the reseller to the client. The client may have several public keys, so the final public key in the verified chain may be any of the public keys of subjects in the provided certificates. As stated earlier, the client has to prove its ownership of these keys during the session establishment. In case that the resolver succeeds in forming a valid certificate chain, it will return the certificates in this chain so that the resource server can create a reduction certificate to be passed back to the client for storing. Creating a reduction certificate means that the resource server issues a new certificate directly to the client, in which the authorization and validity fields are intersections of all the authorization and validity fields in the chain respectively. This optimization means that in general the resolver doesn't have to do heavy processing for long certificate chains every time.

Using the *AccessDecision* object to implement a custom access control policy has one serious disadvantage: when the server invokes the *access_allowed* method, it doesn't pass the parameters of the original method invocation to it. This means that the request parameters still have to be checked by the resource server after the resolver succeeds in finding a valid certificate chain.

7. EVALUATION AND FUTURE WORK

As discussed in section 4.2, the advantages of SPKI certificate based authorization can be summarized as follows: decentralization, delegation of authorization, and support of anonymity when accessing resources. The architecture presented in sections 5 and 6 supports all these features and offers the desired level of granularity; however, there are several things which could be improved.

The first problem has to do with the authorization field of an SPKI certificate. In order to delegate only subsets of permissions, the resolver would have to understand the semantic content of the authorization field, which can only be achieved in very restricted special cases. The easiest, but also most restrictive approach, would be to require that the authorization field stays the same for the whole chain. This would make the job of the resolver much easier, because authorization fields would only have to be tested for equality. Another more sophisticated approach would be to include an additional field to all certificates. This field would contain the address of the server, which could be used to compare the authorization field, i.e., to test for equality and subsets. This solution would need a standard interface, which would take two authorization fields as input parameters and return a code describing the relationship between them. While offering better control of delegation and making the job of the resolver easier, this approach would increase the communication overhead significantly and probably add new security weaknesses to the system.

The second problem concerns the use of the security context object of CORBA to pass the credentials containing the certificates transparently from the client to the server. In our implementation all certificates have to be stored in this context at the client program start up. This is required because we can't always intercept the method calls of the client and add the necessary certificates to the security context on the fly before transmitting the actual call to the server. This kind of client side interception would face the same problem of semantics as the resolver, when it would try to decide which certificates would have to be provided with each call. Storing all certificates in the security context could make it quite large, in case that the client has a lot of certificates stored locally. Even though we use elliptic curve cryptography for keys and signatures in order to save space, each certificate still takes around half a kilobyte in size.

The third problem has to do with the online checks concerning the requests, which the client makes for the resource object. Performing these checks could affect the response time of the services provided by the resource significantly. These kind of checks would have to be performed, for example, in case that the client is entitled to use the resource only a certain number of times. One way to optimize online checks is to make each check return a period of time during which the check doesn't have to be performed [20]. Another more efficient but less secure alternative would be to use a simple session time limit for the connection between the client and the resource.

8. CONCLUSIONS

In this paper we have presented an SPKI certificate based alternative for handling authorization in CORBA based distributed applications. As shown, the CORBA Security Service is very flexible, and is easily extended to handle certificate based authorization. Compared to access control list based approaches, certificates provide much better scalability and flexibility in distributed systems. Privacy protection and delegation of authorizations, both of which become possible with certificates, are also becoming more important as the Internet continues to grow. Despite the problems encountered with certificate based approaches, we believe that certificates show great promise in the field of distributed access control.

REFERENCES

[1] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas and T. Ylönen, Simple Public Key Certificate, Internet-Draft draft-ietf-spki-cert-structure-05.txt, work in progress, Internet Engineering Task Force, March 1998.

[2] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas and T. Ylönen, SPKI Certificate Theory, Internet-Draft draft-ietf-spki-cert-theory-02.txt, work in progress, Internet Engineering Task Force, March 1998.

[3] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas and T. Ylönen, SPKI Examples, Internet-Draft draft-ietf-spki-cert-examples-01.txt, work in progress, Internet Engineering Task Force, March 1998.

[4] D. Maughan, M. Schertler, M. Schneider, J. Turner, Internet Security, Association and Key Management Protocol, Internet-Draft draft-ietf-ipsec-isakmp-10.txt, work in progress, Internet Engineering Task Force, July 1998.

[5] E. Amoroso, Fundamentals of Computer Security Technology, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[7] J. Partanen, P. Nikander, Adding SPKI Certificates to JDK 1.2, Proceedings of the NordSec'98, the Third Nordic Workshop on Secure IT Systems, Trondheim, Norway, November 1998.

[8] OMG, CORBAservices: Common Object Services Specification, December 1999.

[9] OMG, IIOP over SSL specification, February 1997.

[10] OMG, The Common Object Request Broker: Architecture and Specification, July 1998.

[11] P. Nikander, J. Partanen, Distributed Policy Management for Java 1.2, Proceedings of Network and Distributed System Security Symposium, February 4-5, 1999, San Diego, California.

[12] P. Nikander, Y. Kortesniemi, J. Partanen, Preserving Privacy with Certificates in Distributed Delegation, Proceedings of 1999 International workshop on Practice and Theory in Public Key Cryptography, March 1-3, 1999, Kamakura, Japan.

[13] R. Atkinson, Security Architecture for the Internet Protocol, RFC 1825, August 1995.

[14] Recommendation X.500, The Directory: Overview of concepts, models and service, ITU-T, 1993.

[15] Recommendation X.509, The Directory Authentication Framework, volume VIII of CCITT Blue Book. CCITT, 1988.

[16] R. Orfali, D. Harkey, J. Edwards. Instant CORBA, John Wiley & Sons, 1997.

[17] T. Hasu, Storage and retrieval of SPKI certificates using the DNS, Master's thesis, Helsinki University of Technology, Telecommunication Software and Multimedia Laboratory, Otaniemi, Finland, April 1999.

[18] T. Hasu, Y. Kortesniemi, Implementing an SPKI Certificate Repository within the DNS, Unpublished manuscript, Helsinki University of Technology, Telecommunication Software and Multimedia Laboratory, Otaniemi, Finland, August 1999.

[19] W. Yeong, T. Howes, S. Kille, Lightweight Directory Access Protocol, RFC 1777, March 1995.

[20] Y. Kortesniemi, T. Hasu, J. Partanen, A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems, to appear in Proceedings of the 2000 Network and Distributed Systems Security Symposium, San Diego, California, Internet Society, February 2000.