Secure remote management

S. N. Bhatti, G. Knight
Department of Computer Science, University College London, Gower Street, London WC1E 6BT, England, UK
saleem@cs.ucl.ac.uk, knight@cs.ucl.ac.uk
and
D. Gurle, P. Rodier
CNET, France Telecom, 905 Rue Albert Einstein, 06921 Sophia Antipolis Cedex, France
gurle@sophia.cnet.fr, rodier@sophia.cnet.fr

Abstract

Much of the network management technology today still centres around a remote monitoring approach. One would like to have a more intrusive management capability but in a large distributed system one must have confidence that management activities can not be subverted, whether by accident or by malicious intent. To achieve this goal, one requires the management applications to have security mechanisms that will prevent unprivileged users from altering the system accidentally but also, more importantly, to prevent possible attacks from a third party who may disrupt or misuse services. This paper describes some services and mechanisms with which the authors have experimented to allow secure remote management of a distributed system in a real service environment. Although there are many standards documents describing various security mechanisms, some aspects of these documents are not stable and in other cases we can not apply the mechanisms they describe due to restrictions in our development and deployment environment. In such cases we have had to make some adaptations.

Keywords

Network Management, Security Management, Distributed Systems Management.

1 INTRODUCTION

The provision of secure management facilities for distributed applications is very important if the applications operate in an environment that is geographically widely dispersed, operating over a mixture of private and public networks. In general, one must assume that such underlying networks are insecure; that management information may be destroyed or stolen; that malicious third-parties may be able to gain access to the networks and disrupt management activities in a variety of ways. In such cases, the management and security facilities we require must be placed in the parts of the system we can trust – in the applications themselves.

Part of the motivation in the development of the security services described in this document is that they will be deployed in a real service environment, namely in the management of a large X.400(84) [X.400, 1984] mail network.

The mail network also uses a X.500(88) [X.500, 1988] directory service. There are two prime considerations:

- **Protecting existing security mechanisms** If applications are to be controlled remotely then there is a danger that any security services native to the application may be undermined by a third party through subversion of the management exchange. In the particular case of X.400, for example, a range of services including confidentiality, non-repudiation etc, are already provided, so subversion of management control could allow, for example, a third-party to cause mail for one user to be delivered to another. In fact, the native security services will be only as strong as the security services that are provided for management.
- **Deployment in a real service environment** We must develop technology which can be deployed commercially in the short to medium term. Consequently we have sought security solutions which can readily be introduced into today's OSI environment.

1.1 Contents of this paper

The aim of this paper is to give a detailed description of security services and mechanisms we have implemented and the APIs which have been introduced. The analysis behind our choices of security services and mechanisms is summarised in Section 2; a much fuller description of them is given in [Knight et al, 1994]. Section 3 discusses the mechanisms we used and some of the restrictions that affected our design decisions. The implementation of the mechanisms is described in Section 4 with a short summary in Section 5.

2 STATE OF THE ART

The issues of providing security service have received much attention recently, and the the activities of various standards bodies and consortia have resulted in a number of relevant documents. There has been work from ISO and the ITU [X.509, 1988] [X.511, 1988] [X.800, 1991] [CD10183.2, 1992] which includes specific access control features for the purposes of management [CD10164-9, 1992]. This is of particular relevance to the work described in this document which is to be applied in an OSI environment. The ISO mechanisms rely on an infrastructure that uses the RSA system of Public Keys [Rivest et al, 1978] distributed by use of the X.500 directory service [X.500, 1988].

A set of standards that are still maturing but may be extremely important are the Generic Upper Layers Security (GULS) documents [GULS, 1992].

The X.500 documents are stable and use of the infrastructures they describe has been successfully demonstrated in the PASSWORD project [Kirstein et al, 1992]. The ISO/ITU work documents on access control are not fully stable (in the final draft stages). However, they contain mechanisms that were considered to be useful and will probably be present in the final standards, hence it was decided to proceed with their use. The GULS work was not considered sufficiently stable to implement.

A discussion with respect to security **threats** that an application may be subject to, the requirements of the security **services** that are to be implemented, and the **mechanisms** that can be used to realise the services can be found in [Knight et al, 1994].

In this paper, we identify **managed** (server or agent) and **managing** (client or manager) roles for our applications, and we consider a third party that tries to subvert that exchange of information between remote entities in managed and managing roles. Management applications use the OSI Common Management Information Service (CMIS) [CMIS, 1990] [CMIP, 1990].

3 SECURITY MECHANISMS

Many of the mechanisms which implement the security services we require are based on encryption techniques. In choosing mechanisms we have tried to follow the pattern which prevails in the OSI world but, at the same time, to borrow from the other work (such as that for SNMPv2 [Case et al, 1993]) which is geared particularly to the needs of management. The principle difference between OSI and SNMPv2 management services is that the OSI one establishes a long-term, reliable association whilst SNMPv2 does not. This has some impact when security mechanisms are considered for use:

- Confidentiality and integrity mechanisms typically require the two communicating parties to have shared knowledge of a secret value. Without an association it is usual to expect this secret to be known to the two parties *a priori* and it must be stored securely by each of them ready for use – this is what happens in SNMPv2. With an association it is natural to negotiate a new secret value when an association is established thus eliminating the need for secure storage.
- The OSI protocols employed maintain an association that guarantees sequenced delivery of PDUs with very high probability. Further, each PDU has an `invokeID` field – an integer which we can insist must take values from a known sequence. This greatly simplifies the design of a stream integrity mechanism. To achieve the same with SNMPv2 requires a rather complex shared clock mechanism.
- Once an association has been established it will normally be held for a comparatively long period. This makes it reasonable to implement quite complex security mechanisms in the association establishment phase in the knowledge that they will be used only rarely. It is feasible, for example, to use Public Key encryption in association establishment.

With these considerations in mind, the following mechanisms and services were chosen:

- **authenticated associations** To authenticate associations through the use of Public Key encryption using the RSA algorithm. This is the mechanism described in [X.509, 1988] and [X.511, 1988].
- **integrity checks** To add cryptographic checksums to all management PDUs calculated according to the MD5 [Rivest, 1992] algorithm.
- **sequence numbers** To use a well-known sequence for the values of the `invokeID` in ROS PDUs [ROS, 1989].
- **confidentiality** To use secret key encryption in the form of the Data Encryption Standard (DES) [DES, 1988] for protecting confidential data.
- **access control** To implement access control as per [CD10164-9, 1992].

Initial investigations of these mechanisms are reported in [Knight et al, 1994].

3.1 Authenticated associations

For providing mutual peer authentication, the communicating parties exchange credentials based on the X.509 authentication framework and syntaxes defined in the X.511 Directory protocol. When an application (initiator) wishes to establish an association with a peer (responder), it first constructs credentials which consist of the following information:

- **user certificate** This certificate contains the initiators identity cryptographically signed by a Certification Authority (CA) in accordance with X.509.

- **recipient identity** The responder may be able to assume many identities so the initiator provides a Distinguished Name (DN) informing the responder which identity it expects.
- **session key** A secret value that will be used by the mechanism for protecting the PDUs sent on the association. The session key is encrypted using the recipients Public Key before transmission.

Both the encrypted session key value and the recipient DN value are signed by the sender to ensure that they can not be tampered with by a third party. It would have been more convenient to carry this information in the user certificate, but this is not possible due to the certificate's syntax.

An ASN.1 syntax called `SessionCredential` is used to carry the information listed above. We use the `StrongCredentials` syntax [X.511, 1988] for the user certificate and our own `Session-Key` syntax. (At moment, we use the `SessionKey` value for protecting PDUs only – see note on the implementation of confidentiality later). We also use the ASN.1 macros `SIGNED` and `ENCRYPTED` [X.509, 1988].

The `SessionCredential` is sent in the `userInfo` parameter of `CMIPUserInfo` syntax (which is in turn passed to the peer as the `user-information` parameter of the AARQ PDU [ACSE, 1992]).

Authenticating an association is a comparatively expensive operation since the RSA algorithm is complex and we must implement it in software; therefore we perform this authentication just once at association set-up time. Integrity checks – which are relatively cheap – are then applied to all subsequent PDUs sent on the association. In this way we obtain a strong assurance about the origin of PDUs on an association. The authenticated identity can also be used as input to access control decisions.

In fact, we restrict the identity to be a X.500 Distinguished Name (DN); ie. a distinguished name of an entry in the global X.500 Directory. This guarantees that identities are globally unique and is well-suited to authenticating entities such as applications, people, etc.

Before any authentication can take place, an entity must establish its right to assume a particular identity and obtain the corresponding secret key. How this is done is a purely local matter; for example, use of **smart-card** technology or simply the use of a UNIX **filestore** where the secret key information is associated to UNIX **userids** on the client system.

When the responder replies, it sends its own certificate.

3.2 Integrity checks and sequence numbers

These two mechanisms are presented together as their realisation is closely linked. For each PDU sent, the integrity check is evaluated as follows:

1. **Encode PDU using DER to produce byte stream** The CMIP PDU is encoded using the Distinguished Encoding Rules (DER) [X.509, 1988] for ASN.1 to produce a byte stream B. DER ensures that the 'shortest' BER encoding is always used.
2. **Evaluate MD5 checksum for byte stream and session key** The byte stream, B, has the session key value, k, appended to it and the MD5 value for the resulting byte stream B_s is used as the input to the MD5 algorithm, resulting in an 128bit checksum, c:

$$B_s = B * k \tag{1}$$

$$c = MD5(B_s) \tag{2}$$

where the $*$ operator results in a byte stream that is the concatenation of its byte stream arguments.

We carry the value of the checksum outside the CMIP PDU itself as the value of the ROS `invokeID`. However, before the checksum can be used as the `invokeID` value, we must try and ensure that any outstanding `invokeID` values ona an association are unique. As there is no mandatory parameter in CMIP PDUs that will guarantee that during a single association the PDUs (and so the DER byte stream) will be unique, the same checksum will be produced for those PDUs that are the same even if the generation of the CMIP PDUs is separated in time; this provides a potential attacker the opportunity to replay that PDU. Uniqueness is achieved by using a generated sequence of numbers which are combined with the checksum value. The sequence of numbers is generated by using a seed from the session key, so only the initiator and responder can be aware of the sequence. The sequence numbers are used to form the `invokeID` as show by equation 3 and decoded by the receiver using equation 4.

$$i = f(n, c) \tag{3}$$

$$c = g(i, n) \tag{4}$$

where:

- i is the eventual value of the ROS `invokeID`
- n is one of the numbers in our generated sequence of numbers
- c is the checksum value for the CMIP PDU
- f is used by the sender of the CMIP PDU
- g is used by the receiver of the CMIP PDU

Here, the values of n form a known sequence. CMIP and the OSI upper layers provide ordered delivery of PDUs, so we can use such a sequence number mechanism with confidence. As the receiver of the PDU knows n, it can evaluate c for the received CMIP PDU locally and compare it with the received value of the `invokeID` of the ROS PDU.

As there may be many n outstanding, all replies to an initiator request will have to be checked against all these values of n using g. Therefore, the functions f and g should not be computationally expensive, in order to maintain performance.

We have chosen to use the exclusive-OR function for f and its inverse for g. For n, we use the sequence of numbers from a pseudo random number generator. The seed for the generator is taken from the session key value exchanged. As clients and agents communicate asynchronously (in general), there are actually two sequences, one directed from an application that is the initiator of the association (the **initiator-sequence**) and one directed in the opposite direction (the **responder-sequence**). An important property of the number generator we decide to use is that we know its period, so we can ensure that we can always uniquely identify PDUs.

There is a possible weakness in this method which is explained below.

Possible method of attack to the integrity check mechanism

The method of using sequence numbers generated from the session key relies on the fact that the sequence numbers can not be evaluated without knowledge of the session key (which was transmitted encrypted using the recipients Public Key and signed by the sender). However, as the PDUs maybe sent in plaintext in the absence of a confidentiality mechanism, a potential attacker can see when identical PDUs are sent and knows that the only difference in the ROS `invokeID` is that there is a different sequence number. So, it would theoretically be possible (though very computationally expensive) for an attacker who has knowledge of the function f, to work out the sequence and so possibly deduce the value of the session key. While no analysis has been conducted to evaluate the potential for such an attack, we feel that it would be difficult to

perform. Further, we advocate the use of the optional `currentTime` field in CMIP reply PDUs to further deter an attacker. We feel that the use of the `currentTime` field with a granularity of 0.001 seconds (or greater, if possible) in most practical cases would deter such an attack, as this would result in different byte streams for PDUs that might otherwise be identical.

However, as the CMIP `currentTime` field is only available in replies and is optional, we can not insist on or guarantee its use in all cases. Therefore, another solution to evaluating the checksum value from the PDU byte stream may be as follows:

$$B_{sn} = B * k * n_t \tag{5}$$

$$c = MD5(B_{sn}) \tag{6}$$

$$i = c \tag{7}$$

where n_t is the two's complement representation of the number n in the least whole number of bytes.

In this case it would be sufficient for n to be part of a monotonically increasing sequence. The drawback with this solution is, however, that it may require the receiver of replies to perform many calculations of B_{sn} and c if there are many outstanding n, which would affect performance greatly.

3.3 Confidentiality

To prevent unauthorised persons inspecting the contents of a PDU, we can encrypt the bytes that make up the CMIP PDU or the ROS PDU. One method for this would be to negotiate a new transfer syntax for the encrypted encoding of PDUs. However, this may not be possible, for instance if we have bought a stack from a vendor that does not support our encryption transfer syntax. Indeed, this is the case in our service environment and so this solution is not desirable. Instead, we require some application level mechanism rather than a presentation layer mechanism to allow us to send encrypted data.

Moreover, encryption (unless supported by hardware) can be quite computationally expensive and we are sensitive to the general requirement that management operations should not noticeably affect the performance of the systems they are managing. Also, we may not require the encryption of the whole PDU, just certain fields that carry sensitive information. For instance, for a CMIS M-Get request, we may not care that a third part is able to inspect the replies and determine that they are indeed replies to a M-Get request, but we would like to prevent disclosure of the attribute values.

The design of our confidentiality mechanism revolves around the use of ASN.1 macros that embody the encryption and decryption process, converting between an encrypted byte stream and 'wrapper' syntax that carries the encrypted byte stream. The wrapper syntax allows us to selectively encrypt certain fields of a CMIP PDU without modifying the syntax of the PDU. The use of an encryption algorithm and any data associated with the use of the algorithm is notified at association set-up, and indeed the session key value could be used.

When the first experiments were conducted on implementation of the described mechanism, it was decided to use DES. As there was no hardware available to us, we had to rely on software implementations of DES. Our own software implementation achieved an approximate throughput of 0.75 Mb/s (on a Sun4 IPC) and introduced noticeable additional load on the host machine. These constraints were considered unacceptable to allow deployment of the mechanism in our service environment.

Given the fact that confidentiality was not identified as a high-priority security service for our demonstrator and the likely impact on performance of software-based encryption we have not yet proceeded with a full implementation. In the remainder of this paper we concentrate on the services which have been implemented; authentication, integrity and access control.

3.4 Access control

Initially, the access control work at UCL used a mechanism based on security labels and is described in [Knight et al, 1994]. This relied on the use of DNs rather than labels, and has now been supplemented by an access control list scheme implemented by CNET.

ISO has standardized in [CD10183.2, 1992], generic access control models and procedures and in [CD10164-9, 1992] has applied these principles to OSI management. We use these documents and draw from [OMNIPoint016, 1992] to specify and implement our access control service based on an access control list (ACL) scheme. To provide an access control service, we adopt the scenario that a user of a client management application is trying to operate upon some information held at the agent application, and that the identity of this user has already been authenticated during association set-up, and that this authenticated identity information is available to the access control mechanism.

In the context of OSI systems management, control of access to management information may be required in each of the following cases [OMNIPoint016, 1992]:

- association establishment
- request to perform a management operation
- the forwarding of notifications as EventReports

We describe hereafter access control as applied only to the first of these three. An access control mechanism *is* applied to management requests, but it is not based on the use of any additional information received with the request, relying only on the fact that request was received on a secure, authenticated association. Access control for EventReport emission is not applied for the reasons exposed in [OMNIPoint016, 1992]. Also, we separate the activities required to offer access control into **OSI management** activities involving operations upon management information that relate to access control, and **operational** activities which are specific to the access control mechanism which acts upon the access control information. To explain; access control is achieved by passing access control information (ACI) to an access control decision function (ADF). The management activities required to initialise or maintain access control information for each of the items listed above would be very similar, but the operational activities that take place to apply this information will be different when considering each item, i.e. the access control decision function will behave differently.

Access control policy representation

The basic building block of OSI management is the Managed Object (MO). The MO is an abstract representation of a resource that is to be managed. OSI management essentially revolves around the remote manipulation of the attributes, actions and notifications of MOs. Our ACL based scheme is modeled according to the managed object classes defined in [CD10164-9, 1992]. As the policies are themselves realised as managed objects, the same CMIP operations can be used to control and manage the user selected policies. ACI is part of a policy and is expressed as the attributes of a MO. The policies are represented as a set of **rules**. For each policy there may be a set of **global** and **default** rules which can **deny** or **grant** access. The access control information that is used as input to the access control decision function is defined as:

- **access control rules** According to the access control policy representation in use, we use attributes of managed object classes `accessControlRules`, `globalRules` and `default-Rule` to represent access control rules.
- **initiator-bound ACI** This is the access control information provided by the initiator of a management request. In our case, we rely on the distinguished name that was authenticated at association set-up.

- **target-bound ACI** This access control information identifies the management information on which operations are to be performed. In our case, this is the given by the distinguished name of the user and CMIP parameters that identify the MO instances that are to be operated upon, e.g. `managedObjectIntstance`, `scope`, etc.

Both the initiator-bound ACI and the target-bound ACI could also use information that is sent on a per request basis in the `accessControl` field of a CMIP PDU. For our identity based ACL scheme, however, it is sufficient to use the DN of the user: we have confidence that the DN is genuine as it has been authenticated and we are also aware that the request PDU itself has been verified by the integrity mechanisms.

When expressing access control policy the user must state precisely the level of granularity that is required. Although [CD10164-9, 1992] allows very fine granularity we limit the access control to the coarsest granularity, applying it effectively to the association. The reasons for this are mainly concerned with performance and are discussed in [Knight et al, 1994].

Use of access control information

We use the managed object class `aclInitiators` to store target-bound ACI. We use a DN that can be matched against the DN that was authenticated during association set-up. A particular `aclInitiators` instance is identified by a `globalRules` object instance that contains a list of `aclInitiators` and the permissions which apply. The `globalRules` object can grant or deny permissions so with the granularity we have chosen for our access control, we have effectively an all-or-none approach to allowing access to the managed system.

4 DESCRIPTION OF IMPLEMENTATION

Authentication services are provided by the OSISEC package [OSISEC, 1993]. The CMIP implementation is provided by the MSAP library that is part of the OSIMIS [OSIMIS, 1993] management platform. Upper layer OSI services are provided by the ISODE [ISODE, 1991] package. All these packages are implemented in C and C++, running under UNIX type operating systems. Our CMIS/P library is called MSAP.

4.1 Authenticated associations

The authentication information is passed to the peer in the CMIP `userInfo` field. This is passed in the ASN.1 type `EXTERNAL` represented by the MSAP type `External` to the MSAP library calls.

Although there is a complex exchange of information between applications, the information that MSAP user may need to supply in order to set up a secure association is fairly simple. A MSAP user is provided with the following interface to pass information about the secure association to the MSAP library:

```
typedef struct AuthAssocIntegrityInfo_s {
    char *name;          /* String DN of user/application           */
    char *peer;          /* String DN of peer user/application      */
    char *ca;            /* String DN of Certification Authority    */
    char *dsa;           /* Name or ISODE-format address of DSA     */
    char sessionKey[8];  /* MD5 Key - zeros if no key for this assoc */
} AuthAssocIntegrityInfo;
```

The elements `name`, `peer` and `ca` take the form of a human readable DN to identify the user (or user application), the peer (or the peer application) and the Certification Authority (CA) that has signed their credentials, respectively. An example is of a human readable DN is:

```
"c=GB@o=University College London@ou=Computer Science@cn=Saleem Bhatti"
```

The `dsa` is the name of your local X.500 DSA. The `sessionKey` element is the shared secret that will be used to create unforgeable MD5 checksums, as the seed for the random number sequences for the PDUs and also as the DES key. Mapping between this data structure and the ASN.1 `EXTERNAL` representation is provided by the following simple API:

```
int makeAcseUserInfo(AuthAssocIntegrityInfo *info, External **external);
int getAcseUserInfo(External *external, AuthAssocIntegrityInfo **info);
```

4.2 Integrity checks and sequence numbers for CMIP PDUs

There are three aspects to the implementation of the integrity check:

- **Managing the session key values** A new session key must be generated for each association. Knowledge of the session key is required to generate the integrity checks for the CMIP PDUs. The session key information must be accessed by a separate 'sessionKey-manager' function.
- **The MD5 algorithm and checksum generation** The implementation of the MD5 algorithm is taken from RFC1321 [Rivest, 1992].
- **Generating sequence numbers for an association** Each of the two sequence number flows is generated by an 'ID-manager' function.

Managing the session keys

A new session key must be generated for each new association that is set up. To allow this, the following API allows access to a sessionKey-manager function:

```
char *makeMd5Key();
int setMd5Key(const int fd, const char *key);
char *getMd5Key(const int fd);
```

The function `makeMd5Key()` generates a random key value which can be copied to the `sessionKey` element of the `AuthAssocIntegrityInfo` structure. A call to `setMd5Key()` registers the key for use. Both the MSAP library and the MSAP user may then use `getMd5Key()` to access the key value for the association with file descriptor `fd`.

Making the MD5 checksum for a PDU

The API for obtaining the MD5 checksum value for a CMIP PDU is as follows:

```
int makeMd5Value(const int fd, PE pdu, MD5Value *check);
```

`makeMd5Value()` effectively implements equations 1 and 2. The session key value for the association identified by `fd` is found by `makeMd5Value()` by interrogating the sessionKey-manager function. In our implementation, the checksum, c, does not have to be evaluated by the user of MSAP for CMIP PDUs being sent; this is automatically done in the MSAP library when the PDU is generated before being passed down to ROSE.

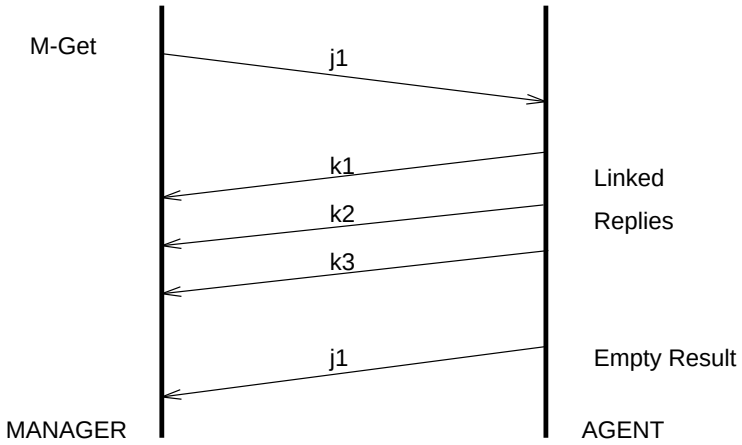

Figure 1 Example showing use of sequence numbers for CMIP PDUs

Generating sequence numbers for the PDUs

For our integrity check mechanism, we are using the mechanism identified in equations 3 and 4. So far we have described the implementation of the functions f and g as well as the generation of c. It remains to describe the generation of n. The sequence is generated by the use of the pseudo random number generator `random(3)` with a state size of 32 bytes. `random(3)` implements a non-linear additive feedback random number generator which has good randomness properties. A user of the MSAP library must generate the next number in the sequence and then pass this as one of the parameters to the MSAP function calls. Should a user wish to generate part of the sequence and then return to a previous point, P, in the sequence, the user is provided with functions to save the state of the random number generator at point P and then restore this state so restarting the random number generator at P.

Use of n and i for applications in a manager role

In the following text the subscript m identifies values associated with a manager application and the subscript a identifies values associated with an agent application. For a request identified by the pair $\{n_m, i_m\}$, the reply PDUs will use values for n as follows: a single CMIP reply PDU, or an EmptyResult PDU will use the n_m value from the original request, and any other PDU will use n_a, part of the responder-sequence. When a manager wishes to issue a M-CancelGet, i_m is used to identify the M-Get to be cancelled.

Where a CMIP PDU is not created, the MSAP library creates the bytes stream B from the DER encoding of n values. In the case of an 'empty' PDU, where the PDU is an EmptyResult, the byte stream B is the DER encoding of n_m and in other cases it is formed from the DER encoding of the next value in the agent sequence n_a.

This is best illustrated with an example. We show a M-Get request that results in the generation of linked replies in Figure 1. Each arrow represents a PDU. The number on top of an arrow is the value of n used to evaluate i for that PDU. The `j` numbers are successive numbers in the n_m sequence, and the `k` numbers are successive numbers in the n_a sequence.

The numbers in the sequences are generated by a simple API:

```
void setIRStatus(const int fd, const IRStatus s);
int makeId(const int fd);
int makePeerId(const int fd);
```

The association file descriptor, `fd`, a call to `setIRStatus()` with registers whether the applications is the initiator or responder for that association. This will allow generation of the two

number sequences using `makeId()` and `makePeerId()`.

4.3 Access control

An initial set of `aclInitiators` objects are created and initialised at agent start-up. These objects are always present and can be used to control not only access to the management information held at the agent, but also control the modification of access control information, e.g. creation of new access control objects or modification of existing ones.

Location of the access control operational activities

The standards leave open the location of access control elements that perform the access control decision function. They can be treated as being logically distributed between the manager and agent systems. However, in OSIMIS the access control decision function is built into the agent software (i.e. local and centralised), and this is in keeping with the way in which the mechanism with which the identity of communicating parties is authenticated within our model. The manager system must always authenticate itself to the agent, and the agent will always confirm the association by sending its own credentials, but the manager application does not have to authenticate the agent if it so desires.

The access control decision function is represented by an instance of an `accessControlRules` managed object, and there is only one instance of this object in the management information tree. It represents the access control decision function. The following access control information is located in the agent system:

- **policy-ACI** This is represented by the `defaultAccess` and `denialGranularity` attributes of the `accessControlRules` object class and the `globalRules` object class. Instances of the `globalRules` objects exists to deny and grant access. Each one identifies, by the value of its `initiatorList` attribute, a list of other managed objects to whom access is granted or denied. The deny lists are checked before the grant lists and there is a `defaultAccess` attribute used when an identity is not matched. The security manager can allow or deny access to CMIS services by modifying this attribute for the unknown users. Thus we effectively have **privileged** users, **normal** users and **restricted** users.
- **target-ACI** This is represented by instances of the `aclInitiators` managed object class which identifies a set of peers by their DN. There are two instances of this class; one which is associated with the list of restricted users (the deny list) and the other that is associated with the list of privileged users (the grant list).

The object instances are part of the management information tree of the agent, and so can be operated upon just like other managed object instances using the CMIP primitives to perform management activities.

Performance considerations

To improve the performance of the access control decision function there is a simple (volatile) cache mechanism which caches all the identities of peers that frequently access the agent during the time it is active (**retained ACI**). When a new access request arrives, if the access control information has not been modified, then a simple table look-up for the initiator in this cache speeds up association set-up. This avoids scoping and filtering operations needed to interrogate the information in the object instances to find the permission for the authenticated identity and reduces the time for evaluating the access decision by approximately 50%.

5 SUMMARY AND CONCLUSIONS

The following security mechanisms have now been implemented within the OSIMIS management platform:

- **authentication** using the RSA based Public Key method.
- **sequence numbers** and MD5 **integrity checks** to provide data origin authentication and stream integrity.
- **access control** based on the use of access control lists to provide protection against unauthorized access to management applications and management information.

These are now being evaluated within MIDAS (ESPRIT Project 6331) as part of the final demonstrator.

For computationally expensive encryption there may be additional load introduced on the host machine and so we must seek to use it only where necessary and preferably with the aid of hardware. We find that we are prepared to pay the price of software RSA encryption for authentication at association set-up but providing confidentiality using software for DES is not practical for our service environment. Experiments continue at UCL with our confidentiality mechanism.

For the integrity check and stream integrity mechanism, MD5 is relatively cheap, and we are again prepared to accept this to be implemented in software. We feel that the integrity mechanism described in this paper for the CMIP protocol could be applied to any ROS based protocol.

An access control list scheme is well suited for implementing the access control service we require for managing our X.400 system, providing protection against deliberate attack and accidental misuse. However, implementing partially the `itemRules` and `targets` managed object classes would allow the agent finer granularity for control. However, we do not see a reasonable way of implementing `itemRules` completely within OSIMIS and still maintain performance. The use of the identity based ACL scheme coupled with the integrity schemes for the CMIP PDUs allows us to make access control decisions without requiring further information. This means that we do not incur the additional overhead of processing any per request access control information. Also, the increase in performance is considerable with the introduction of caching.

6 ACKNOWLEDGEMENTS

The work conducted at University College London was partially financed by the MIDAS project under the ESPRIT funding initiative.

7 REFERENCES

[X.400, 1984] CCITT Recommendation X.400, Message Handling Systems: System Model Service Elements, Geneva, 1984.

[Rivest et al, 1978] R. L. Rivest, A. Shamir, L. A. Adleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, Communications of the ACM, number 21, volume 2, pages 120 - 126, February 1978

[X.500, 1988] CCITT Recommendation X.500, The Directory – Overview of Concepts, Models and Services, Geneva, March 1988.

[X.509, 1988] CCITT Recommendation X.509, The Directory – Authentication Framework, Geneva, March 1988.

[X.511, 1988] CCITT Recommendation X.511, The Directory – Abstract Service Definition, Geneva, March 1988.

[X.800, 1991] CCITT Recommendation X.800, Security Architecture for Open Systems Interconnection for CCITT Applications, Geneva, 1991

[CD10183.2, 1992] ISO/IEC CD 10183.2, Information Technology – Open Systems Interconnection – Security Frameworks in Open Systems – Part 3: Access Control, 16 June 1992.

[CD10164-9, 1992] ISO/IEC CD 10164-9.3, Information Technology – Open Systems Interconnection – Systems Management – Part 9: Objects and attributes for Access Control, Borehamwood, UK, December 1992.

[GULS, 1992] ISO/IEC CD 11586, Information Technology – Open Systems Interconnection – Generic Upper Layers Security, December 1992.

[Kirstein et al, 1992] P. T. Kirstein, P. Williams, Piloting Authentication and Security Services Within OSI applications for R&D information (PASSWORD), UCL Department of Computer Science, April 1992.

[Case et al, 1993] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, Introduction to version 2 of the Internet-standard Network Management Framework, Internet RFC 1441, April 1993.

[OMNIPoint016, 1992] Network Management Forum, Application Services: Security of Management, OMNIPoint/NM-Forum 016, Bernardsville, NJ, August 1992.

[Rivest, 1992] R. Rivest, The MD5 Message-Digest Algorithm, Internet RFC 1321, 16 March 1992.

[ROS, 1989] ISO/IEC 9072, Information processing systems – Text Communication – Remote Operations, 1989.

[DES, 1988] National Institute of Standards and Technology, Data Encryption Standard, FIPS Publication 46-1, January 1988.

[Knight et al, 1994] G. Knight, S. Bhatti, L. Deri, Secure Remote Management in the ESPRIT MIDAS project, Proceedings of IFIP WG 6.5 International working Conference on Upper Layer Protocols, Architectures and Applications, Barcelona, June 1994

[ACSE, 1992] CCITT Recommendation X.227, Connection Oriented Protocol Specification for the Association Control Service Element, September 1992.

[CMIS, 1990] ISO/IEC 9595, Information technology – Open Systems Interconnection – Common management information service definition, May 1990.

[CMIP, 1990] ISO/IEC 9596, Information technology – Open Systems Interconnection – Common management information protocol specification, May 1990.

[OSISEC, 1993] UCL Department of Computer Science, The OSI Security Package OSISEC User's Manual, May 1993.

[OSIMIS, 1993] UCL Department of Computer Science, The OSI Management Information Service User's Manual, Version 1.0 for system version 3.0, February 1993.

[ISODE, 1991] UCL Department of Computer Science, The ISODE User's Manual, Version 7.0, July 1991.

8 BIOGRAPHIES

Saleem N. Bhatti received a B.Eng.(Hons) in *Electronic and Electrical Engineering* in 1990 and a M.Sc. in *Data Communication Networks and Distributed Systems* in 1991, both from

University College London. Since October 1991 he has been a member of the Research Staff in the Department of Computer Science, involved in various communications related projects. He has worked particularly on Network and Distributed Systems management.

Graham Knight received his M.Sc. from UCL in 1980 and has since worked in the Computer Science department as a researcher and teacher. He is now a Senior Lecturer and has led a number of research efforts in the department. These have been concerned mainly with two areas; network management and ISDN. These interests have been pursued through three ESPRIT projects; INCA, PROOF and MIDAS. The network management activities have led ultimately to the OSIMIS management platform whilst the ISDN activities have resulted in the design, production and ultimate deployment of the UCL Primary Rate ISDN gateway.

David Gurle received his M.Sc. in *Computer Science and Telecommunications* in 1992 from Ecole Superieure d'Ingenieurs en Genie des Telecommunications et en Informatique (Paris – Fontainebleau). He worked for one year in Digital on CORBA and Intelligent Networks before joining CNET in 1993. Since then, he has worked on network and distributed systems management.

Philippe Rodier received his engineering degree in *Mechanical Sciences* in 1978 from Insitut National des Sciences Appliquees (Lyon). He worked for four years in Thomson CSF. He than worked for five years in Texas Instruments. He received his M.Sc. in *Computer Science* in 1988 from Cerics. Since 1988 he has worked in CNET and since 1992 he leads a group which focuses on applications of computing to network management.