Secure management information exchange

Saleem N. Bhatti
S.Bhatti@cs.ucl.ac.uk
+44 171 419 3249

Kevin M. T. McCarthy
K.McCarthy@cs.ucl.ac.uk
+44 171 419 3687

Graham Knight
G.Knight@cs.ucl.ac.uk
+44 171 380 7366

George Pavlou
G.Pavlou@cs.ucl.ac.uk
+44 171 380 7215

Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom

16 August 1995

Abstract

This paper describes the design and implementation of a secure management protocol for the management of distributed applications. The protocol is a modified use of the ISO CMIP protocol, with additional mechanisms and behaviour to provide the following security services:

- **Mutual authentication** of communicating parties. Both parties can prove to each other that they are who they claim to be by the exchange of signed credentials.
- **Stream integrity** for management information packets (**protocol data units – PDUS**). The management information exchanged between the parties is protected from replay, misordering, modification, insertion and deletion of the PDUs.
- **Confidentiality** of the management PDUs. Only the communicating parties can read the information passed between them. The mechanism used also provides **back traffic protection** and **perfect forward secrecy**.

In previous work we have implemented a public-key based system. Here, we present an experiment based on the use of a secret-key mechanism, for a faster, **lightweight** approach. The authentication mechanism makes use of the MD5 algorithm and the DES encryption standard. The PDU integrity mechanisms make use of a pseudo random number sequence for PDU numbering and the MD5 algorithm for generating unforgeable signatures for the PDUs.

A discussion making comparisons with the public-key approach, plus suggestions for improvements and further work is included.

Keywords

network security, network management, distributed systems security

1 Introduction

To establish management facilities in an open environment, it is of paramount importance that we have confidence in the operation of the management applications. we need to be sure that they can perform the tasks they are required to do and that no other applications can perform or subvert these tasks. This is particularly important if the management applications and the systems being managed are geographically widely dispersed, so that a possible attack directed at normal management operations is physically hard to counter.

In management of network elements, we can often use physical security to prevent some attacks, e.g. allowing restricted access to switches and routers etc. However, this may mean that remote management is not possible and that a human administrator must be physicaly in the presence of the equipment to perform management tasks, for instance, by use of a terminal device connected directly to the equipment. Further, if we are to try and manage *distributed applications*, then *remote distributed management applications* will need to cooperate to achieve the management goals. In such an environment, we must have a secure exchange of management information between all the management applications.

In general, we must assume that the underlying communications networks are insecure and that the management applications know best what the security requirements are to perform their particular task. Also, in the case of a distributed application, we must endeavour not to weaken any existing security features that are part of the normal behaviour of the application. For instance, if we had a managed e-mail system, then it may be possible for an attacker to cause mail to be delivered to persons other than the intended recipients by subverting the management system in use rather than directly attacking the e-mail system.

This paper describes a lightweight security mechanism which we have implemented for ensuring a secure exchange of management information. For the case of authentication and integrity, we have not needed to modify the syntax of the CMIP protocol, but for providing confidentiality we need to encrypt the CMIP PDU.

The discussion regarding security requirements for applications is normally given in terms of:

- **Threats:** the attacks that one wishes to guard against.

- **Services:** the services that can be employed to counter the threats identified.
- **Mechanisms:** the mechanisms that can be used to implement the services that are required.

In the rest of this paper, the discussion is presented in terms of the management interactions that take place between **managed systems** (a **server** or **agent**) and the **managing system** (a **manager** or **client**). Note that an application may have many managment interactions with other applications and within those interactions it may, in general, take both a managing and managed role during the lifetime of the interaction. Further, we describe the services used and mechanisms employed with in our implementation of the ISO CMIP protocol [1]. As CMIP is a connection oriented protocol, we identify the **initiator** of the connection request (or **association** in OSI parlance) and the **responder** to the connection request.

1.1 Contents of this paper

In section 2 we take a look at the goals for this particular work. In the next three sections we look at, in turn, the particular threats that we wish to protect against (section 3), the security services we intend to use (section 4) and then describe the mechanisms that we have used to realise them (section 5). Section 6 describes our implementations. After a brief discussion in section 7, we round off with a summary in section 8.

2 Security requirements for management applications

A general description of the threats, services and mechanisms is given in the X.800 document [2]. OMNI-Point016 [3] has similar descriptions, but tailored to the context of network management using the ISO management model.

Previously, we have investigated the use of **public-key** systems as the basis of our security system [4, 5, 6]. Such systems offer **strong** assurances of authenticity and confidential exchange of information. There are several authentication frameworks that are based on public-key approaches, the most commonly nown of which are X.509 [7] and PGP [8]. Another system that makes use of the X.500 directory is DASS [9].

Here we consider the use of a **shared-key** system, based on the use of a secret-key value that is shared

between each responder/initiator pair. The system aims for a lightweight approach, with the following goals in mind:

G1 Speed: public-key systems rely on computationally expensive algorithms that can be slow, particularly when implemented in software. Shared-key systems based use of secret-key, block algorithms tend to be much faster.

G2 Independence of operation: public-key systems make use of an asymmetric key-pair, the **private-key** and the **public-key** parts. Although the public-key part of the key can be exchanged out-of-band, current large public-key systems rely on the public-key part being stored as part of credentials in a 'well known' place (e.g. a directory service) and signed by trusted third party (the **certification authority (CA)**), for example, the ITU X.509 system [7]. The use of a shared secret-key removes the need for this extra level of indirection and the communication overhead that may be associated with it. This increases the speed of the system as a whole and reduces its reliance on a secure and reliable directory service.

A further comparison of the public-key and secret-key systems is made in section 7.

3 Security threats

The particular threats that we have considered are as follows:

T1 Masquerade: where an attacker takes a false identity.

T2 Replay: where an attacker replays previously captured management PDUs.

T3 Insertion/Deletion: where an attacker constructs fake PDUs and inserts them into the communication stream, or removes PDUs from the communication stream.

T4 Disclosure: where an attacker can observe sensitive management information by inspecting the contents of the management PDUs.

T5 Denial of service: where an attacker can somehow prevent a management application from offering and/or using a management service.

The Internet community has also been looking at the security problem, and from the IPSEC Working Group [10] and the Network Working Group [11] we find the following threats identified:

T6 Examination of previous traffic: an attacker is able to capture and store PDUs and then read them at a later date, when the security of future network traffic is compromised.

T7 Examination of future traffic: an attacker is able to breach the security of future PDUs when the security of current or previous network traffic is compromised.

T8 Clogging: an attacker is able to tie-up computing resources by making continuous bogus authentication requests to management applications.

T6 and T7 can be seen as special cases of threat T4 (Disclosure) but imply that the techniques used to protect against T4 must be such that there are parameters that can be changed in the mechanism so that T6 and T7 are also countered. For instance, use of a key based mechanism where the key can be changed frequently and easily.

T8 is a special case of threat T5 (Denial of service). Denial of service is the hardest form of attack to counter. Denial of service attacks could range from a simple cutting of the wire(!) to more cunningly devised approaches such as clogging or barrages of fake PDU storms (packet flooding) to swamp the network resources. In the rest of this paper, we do not discuss the countering of denial of service attacks.

4 Security services

To counter the threats listed in the section 3, we have chosen the following services:

S1 Mutual authentication of communicating parties: the initiator and responder exchange credentials at connection set-up. The credentials contain an electronic signature that each party can use to validate the other's identity. This counters threat T1.

S2 PDU sequence numbering: a pseudo random number sequence will be used to number the PDUs. The seed for the pseudo random sequence is exchanged at connection set-up time.

S3 PDU electronic signatures: the sender of the PDU generates a signature for each PDU. The signature is sent with the PDU and verified by the receiver. A secret value that is used in the generation of the signature is exchanged at connection set-up time. Together with S2, this counters threats T2 and T3.

S4 Encryption of CMIP PDUs: the CMIP PDUs will be encrypted. The information to be used in the encryption algorithm (e.g. key) will be exchanged at connection set-up time. This counters threat T4.

(The method of countering threats T6, T7 and T8 are given in section 5.) The use of the services are based around the exchange of **credentials** in the implementation of S1. Both initiator and responder must each produce, independently, a set of credentials that can be validated by the other. The credentials consist of a **token** that contains information about the identity of the sender and any information/parameters required for S2, S3 and S4.

5 Security mechanisms

The mechanisms that are used to implement the services described in section 4 are as listed below:

M1 Exchange of credentials: during connection set-up, the communicating parties send each other a set of credentials that contain an electronic signature. The credentials consist of a **token** and a **signature** for that token; the token contains parameters for the operation of M2, M3 and M4 (explained below). The uniqueness of the signature is guaranteed by the use of a **shared-key** – a secret value known *a priori* only to the two parties in the association set-up. This provides services S1.

M2 Use of well-known PDU sequence numbers: a sequence of numbers is generated using a pseudo random number generator (PRNG). The sequence is 'well-known' to both parties as the seed for the PRNG will be exchanged securely at connection set-up time. This provides service S2.

M3 Electronic signatures for PDUs: strong checksums are generated for each PDU by the the use of a hashing function. The checksum can only be generated by the communicating parties as part of the input to the signature generation function is a secret value exchanged securely at connection

set-up time. (The signature mechanism is very similar to that used for M1.) This provides service S3.

M4 Secret-key encryption of PDU byte stream: the CMIP PDU is encrypted before being transmitted. The key for the encryption function is exchanged securely at connection set-up time. This provides service S4.

The threats T6, T7 and T8 listed above can also be countered by careful selection of the implementation of the mechanisms listed above.

Let us first consider T6 and T7. For T6 we require a facility for **back traffic protection** and for T7 a facility providing **perfect forward secrecy** [10]. Both these facilities can be implemented if we ensure that a new encryption key – a **session-key** – is used for M4 every time a new connection is set up. Further, we generate this key as a random value just before it is required for use, and so we do not need secure secondary storage for it.

An attack in the form of T8 relies on the fact that the authentication procedure is computationally expensive and will tie-up resources at the recipient of authentication requests. A counter measure suggested in [11] is to use a **cookie** that is a hash value formed from, say, the IP/UDP/TCP addressing information of the sender. As will be explained later, we use a lightweight, shared-key based approach for signatures to provide authentication of user/application identities that achieves a similar effect.

Hence, we explicitly counter threats T1 to T5, whilst T6, T7 and T8 are countered as side-effects of the implementations of the mechanisms we have chosen.

5.1 State of the art

There is currently a lot of activity within the Internet community and standards bodies such as the ITU and ISO concerning the formulation of security recommendations for network and distributed systems security.

5.1.1 The Internet community

As well as the work for securing IPv6 [15, 16, 17], the IETF IPSEC working Group have produced an Internet Draft describing the management of security information [10]. The IETF work also addresses similar threats to the ones listed as T1 to T8. Of great relevance here is the considerable effort being directed towards providing a security infrastructure for SNMP. SNMPv1 had no security features, and the original SNMPv2 security recommendations [12] have not been widely accepted by the Internet community. Currently, there is activity not only to revise the SNMPv2 security infrastructure [13, 14], but also to provide security facilities that could be retro-fitted to existing SNMPv1 technology [19].

At the time of writing, the activities in the Internet community centre around the provision of the following security mechanisms for IPv6 and SNMP:

- Management of security related information for IP [10].
- Security architecture for IP [15].
- Authentication of IP packet headers [17].
- Encryption of IP packet payloads [16, 18].
- ('Official', i.e. produced by an IETF Working Group) authentication and encryption for SNMP [13, 14].
- ('Unofficial') authentication only [19], and authentication and encryption [20, 21] for SNMP.

(Additionally, there is access control work for SNMPv2.) Although the IP and SNMP work does not explicitly state that the use of any particular algorithm is mandatory, the (very strong) implication is that the mechanisms are implemented using the MD5 algorithm [22] to produce electronic signatures (SHA [23] is also mentioned) and the DES [24] algorithm (i.e. a secret-key algorithm) to encrypt PDUs.

Although SNMP security is also shared-key/secret-key based, there are two main problems with most of the approaches:

- In general, the stream integrity is achieved by use of 'loseley synchronised' clocks between manager and agent. As only one clock exists at the agent, this could cause problems if the agent is communicating with more than one manager at the same time (the *leap-frogging clocks* problem known to the SNMP community).

- Back traffic protection and perfect forward secrecy (countering T6 and T7) cannot be provided as the same key is used each time for encryption of PDUs. As, there are currently no in-band methods to establish a session-key, SNMP security keys must be changed frequently to prevent possible know-plaintext or traffic analysis attacks on the PDUs.

Neither of these problems exists with CMIP; as it is a connection oriented protocol, during the connection set-up phase, a new session-key can be exchanged to be used in the integrity and encryption mechanisms.

At the time of writing, other major problems appear to exist with the IP security proposals, and there is great discussion within the Internet community [25].

5.1.2 The ITU and ISO

The international standards bodies have also been making efforts to define security features for use in data communications applications. Although the particular area of security for *management protocols* has not been addressed, there is relevant work which could be applied. (This is in keeping with the generality of the **service element (SE)** approach to building application level protocols in the ITU/ISO philosophy.) The current work in this area from the ITU/ISO is the **Generic Upper Layer Security (GULS)** document [26]. This draft standard provides a service definition and protocol for a new application level service element that can provide security services such as authentication and confidentiality for any ISO application level protocol. There are also proposals for a general security framework for open systems [27]. These documents are now in the final stages of becoming full standards.

More mature and stable work is available in the form of the authentication framework that forms part of the ITU/ISO Directory service, documented in X.509 [7] and X.511 [28]. This provides a public-key based approach to authentication and uses the ITU/ISO Directory service as the repository for various security related information including credentials for users and certification authorities (CAs), as well as revocation lists for credentials that are no longer valid. In X.511, syntaxes for credentials are defined that can bear a signature from a CA. This syntax can be used not just by the directory but by any application that uses the ITU/ISO association control service element (ACSE [29]). The authenticated information is a globally unique identity – a **distinguished name (DN)**. An example of a DN is:

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

The identity can refer to a real person or an application (in fact it can be anything). Once a DN has been authenticated it can then be used for other purposes, e.g. as input to access control decision functions. However, an application using the X.509 framework, in general, will need capabilities to access the Directory service to look up:

- Public-key information of other users and applications.
- CA credentials.
- Revocation lists.

Caching Directory information is always possible but at the cost of losing timeliness of updates to such sensative information as revocation lists.

5.2 Authentication

The authentication mechanism we have chosen makes use of DNs. The signature is formed for a token that contains the DN (along with some other information). The authentication mechanism is based on the use of a shared-secret which is known *a priori* to both parties. This is exchanged out-of-band. (The implications of this are discussed later.) The algorithm is described below.

ENCRYPT(K, M)	encrypt message M using key K
DECRYPT(K, M)	decrypt message M using key K
HASH(K, M)	make keyed hash value of message M using key K
RANDOM(p, q)	generate a random byte stream, minimum size p, maximum size q
$D.m$	data member m in data structure D
$M@A \rightarrow B$	A sends message M to B
$A \leftarrow M@B$	A receives message M from B
$C1?A1 : A2$	if $C1$ is true then do $A1$ else do $A2$
$D1 = D2$	assignment
$D1 == D2$	comparison

$A \vdash V$	A knows value V

S	signature	T	token	C	credential
K_{IR}	shared-secret	K_S	session-key	I	initiator
R	responder	DN	distinguished name	E	expiry time
t	validity period				

Subscripts

I	initiator	R	responder

Prerequisites:

$$I \vdash K_{IR} \quad , \quad R \vdash K_{IR} \tag{1}$$

$$I \vdash t \quad , \quad R \vdash t \tag{2}$$

K_{IR} is the shared-key that is known only to I and R. The secrecy of this value is the basis for the establishment of a **trust relationship** between I and R. The value t is a constant value that is effectively the period of time (in seconds) for which the credentials are considered to be valid. (t will typically be about 10 seconds or less.)

At I:

$$K_S = \text{RANDOM}(8,8) \tag{3}$$

$$K = \text{ENCRYPT}(K_{IR}, K_S) \tag{4}$$

$$E_I = \text{now} + t \tag{5}$$

$$T_I = \{DN_I, DN_R, E_I, K\} \tag{6}$$

$$S_I = \text{HASH}(K_{IR}, T_I) \tag{7}$$

$$C_I = \{T_I, S_I\} \tag{8}$$

$$C_I@I \rightarrow R \tag{9}$$

The signature, S_I, prevents the forging of T_I, i.e. prevents a masquerade attack (T1). As, only I and R know the value K_{IR}, only they can generate and/or reproduce a correct value of S_I. To prevent replay (T2) of C_I, the signature S_I is only useful if each separate instance of T_I (and hence S_I) is

different. This is assured by the use of a randomly generated session-key value K_S and by use of the timestamp E_I. K_S effectively takes two roles – that of the session-key that will be used to initialise the integrity and confidentiality mechanisms after connection set-up, and also that of a **nonce**. A nonce is a (usually randomly generated) value to be used only once and then discarded, and serves to add some unpredictability to the contents of the token T_I. (The reason for using a K_S value of length 8 will be explained later.)

At R:

$$R \leftarrow C_I@I \tag{10}$$

$$e = \text{noError} \tag{11}$$

$$S = \text{HASH}(K_{IR}, C_I.T_I) \tag{12}$$

$$S == C_I.S_I \quad ? \quad \text{continue} : e = \text{invalidSignature}, \text{goto \_sendResponse} \tag{13}$$

$$E_I > \text{now} \quad ? \quad \text{continue} : e = \text{credentialsExpired}, \text{goto \_sendResponse} \tag{14}$$

$$T_I.DN_R == DN_R \quad ? \quad \text{continue} : e = \text{unknownRecipient}, \text{goto \_sendResponse} \tag{15}$$

$$K_S = \text{DECRYPT}(C_I.T_I.K, K_{IR}) \tag{16}$$

$$R \vdash K_S \quad ? \quad e = \text{sessionKeyReused} : \text{continue} \tag{17}$$

_sendResponse :

$$E_R = \text{now} + t \tag{18}$$

$$T_R = \{DN_R, e, E_R\} \tag{19}$$

$$S_R = \text{HASH}(K_{IR}, T_R) \tag{20}$$

$$C_R = \{T_R, S_R\} \tag{21}$$

$$C_R@R \rightarrow I \tag{22}$$

$$e == \text{noError} \quad ? \quad \text{continue} : \text{drop connection} \tag{23}$$

R receives T_I and performs several checks on T_I. The failure modes that could result from these checks are:

- **invalidSignature:** the signature could not be verified. This could suggest that the contents of T_I have been changed. Implicit to this step, of course, is the following:

$$R \vdash T_I.DN_I$$

i.e. that the responder knows of the initiator (the prerequisite step 2) so that DN_I can be used to look up K_{IR}.

- **credentialsExpired:** the expiry time recorded in the token has been exceeded. This could be due to an abnormal delay in the credentials being received by the responder, suggesting a possibility that an attacker is trying to intercept the connection request in some way. It could just mean that there was an abnormal amount of network congestion *en-route*(!), so the value of t must be chosen carefully to reflect the network environment in which this scheme is to operate correctly.
- **unknownRecipient:** the responder does not think that it is the intended recipient of the message. Such an error could be caused by an attacker trying to replay a previously captured credential in an attempt to connect to a different responder.
- **unknownInitiator:** the responder does not know the creator of the credential. This is to prevent a similar attack as for unknownRecipient.
- **sessionKeyReused:** the responder has detected that two connections have been attempted with the same session-key. This failure mode prevents an attacker from replaying credentials within the expiry period of those credentials.

The last of these failure modes relies on the fact that the value of K_S is generated in a manner that is likely to yield different values each time. This can be achieved by, for example, using the current time as part of the function that produces the seed for the random number generator used. Recommendations for randomness properties for use in security systems is presented in [30]. The use of a fast hash function for the production of signatures not only achieves G1 and G2, but also counters the clogging attack (T8) in a similar fashion to that recommended in [11] (i.e. it is fast and computationally inexpensive to evaluate). If any of the checks on T_I fail, then the connection set-up is aborted.

Once the credentials have been checked, R has authenticated I and we can say that R trusts that it has set up a connection with I. R must now send a set of credentials back to I so that R can be authenticated by I. As well as R's identity, DN_R, and an expiry timestamp, E, the responder's token, T_R, contains any

error information that may have resulted from R's attempt to process T_I. In the event of a history of several failures in authentication, this error information giving the reason for the failure may be important in tracking down a possible security attack. One may consider it more secure to use the 'fresh', securely exchanged value K_S in generating S_R rather than K_{IR}. However, to notify the error, e, to I, we must use K_{IR} to sign T_R, as R does not yet have confidence that the sender of T_I is indeed I.

At I:

$$I \leftarrow C_R@R \tag{24}$$

$$S = \text{HASH}(K_{IR}, C_R.T_R) \tag{25}$$

$$S == C_R.S_R \quad ? \quad \text{continue : drop connection} \tag{26}$$

$$C_R.e == \text{noError} \quad ? \quad \text{continue : drop connection} \tag{27}$$

$$C_R.E_R > \text{now} \quad ? \quad \text{continue : drop connection} \tag{28}$$

$$I \vdash C_R.DN_R \quad ? \quad \text{continue : drop connection} \tag{29}$$

$$\text{connection established} \tag{30}$$

The responder's credentials are checked, and if the checks succeed then the I and R have mutually authenticated each other and the connection set-up is completed, else it is aborted.

5.2.1 Authenticated connection closure

To prevent the possibility of an attacker sending false connection-close requests, we also authenticate a connection closure at the end of a management communication session. The call closure can be initiated by either the original initiator, I, or the original responder, R, so in the description below we simply refer to them as A and B, the communicating parties. The mechanism is as follows:

G GoodBye token A party A B party B

At A:

$$S_A = \text{HASH}(K_S, DN_A) \tag{31}$$

$$G = \{DN_A, S_A\} \tag{32}$$

$$G@A \rightarrow B \tag{33}$$

At B:

$$B \leftarrow G@A \tag{34}$$

$$S = \text{HASH}(K_S, DN_a) \tag{35}$$

$$S == G.S_A \quad ? \quad \text{drop connection : ignore} \tag{36}$$

$$B \vdash DN_A \quad ? \quad \text{drop connection : ignore} \tag{37}$$

As the signature, S_A, is generated using the the session-key, K_S exchanged securely during the connection set-up, we can be sure that the issuer of the connection-close request is in fact the party with which we originally established the association.

5.3 Confidentiality

The confidentiality mechanism encrypts management PDUs. Padding bytes with random values – **junk** – are used to help protect the encrypted PDUs from traffic analysis or know-plaintext attacks. The mechanism used for creating the encrypted PDU is shown below.

V	an initialisation value	B	byte stream
$JP1$	junk pre-padding for PDU	$JP2$	junk post-padding for PDU
PDU	the original PDU	PDU_E	the encrypted PDU
$Q * P$	Q logically concatenated with P		

$$V = \text{RANDOM}(8, 8) \tag{38}$$

$$JPI = \text{RANDOM}(0, 127) \tag{39}$$

$$JP2 = \text{RANDOM}(0, 127) \tag{40}$$

$$B = V * JP1 * PDU * JP2 \tag{41}$$

$$PDU = \text{ENCRYPT}(K_S, B) \tag{42}$$

The session-key, K_S, exchanged securley at connection set-up is used as the encryption key. As a new value for K_S is exchanged each time a connection is set up, we have protected against T6 and T7.

5.4 Integrity

The integrity mechanism for the PDUs is based on the generation of two different values that are to be carried along with each PDU:

- **PDU sequence number:** each PDU has a sequence number that is generated by the sender of the PDU. The sequence number is part of a pseudo random sequence as generated by a PRNG. The session-key, K_S, is used to seed the PRNG.
- **PDU signatures:** each PDU has a strong checksum evaluated for it. This checksum is generated using the same mechanism as that which is used to generate the signatures used in the authentication mechanism with one difference; the key used for the creating the hash value (signature) is the session-key, K_S, rather than the shared-key, K_{IR}. This ensures that the same traffic generated during different management connections will always have different checksums and so protect against T6 and T7.

Both these checks will need to be carried with the PDU. However, the ROS PDU only has one field, `invokeID`, that can be used. So, the value that is assigned to the `invokeID` is the bitwise exclusive-OR of the sequence number and the checksum value. The mechanism is show below.

C	checksum	N	PDU number sequence
R	the ROS `invokeID`	PRNG(X, n)	nth value in sequence seeded by X

$$C = \text{HASH}(K_S, PDU_E) \tag{43}$$

$$N_n = \text{PRNG}(K_S, n) \tag{44}$$

$$R = C \oplus N_n \tag{45}$$

As the communication between manager and agent is, in general, asynchronous, there are actually two PDU number sequences, one directed from manager to agent and one from agent to manager. Different

parts of K_S are used as the seed for each sequence.

We note that this system of PDU integrity checks also implicitly provides a form of **authentication** and **non-repudiation** service for the management PDUs. This is because only the two communicating parties can possibly generate the correct PDU signatures and PDU sequence numbers. The checksum, C, is additionally protected if the integrity mechanism and confidentiality mechanism are used to gether as C is evaluated for the uncrypted PDU and not PDU_E which is seen 'on the wire'.

6 Implementation

Our implementation is called **MSAP Lightweight Security (MLS)**. MSAP (management service access point) is our implementation of the CMISE. The implementation of the mechanisms is in C and C++ on UNIX based systems.

We use the following pieces of software:

- The OSIMIS package [32] to provide the ISO management infrastructure and CMIP protocol implementation.
- The ISODE package [33] to provide the upper layer ISO communications stack.
- The MD5 implementation in RFC1321 [22].
- An implementation of DES written at UCL.

The value of K_S is used as the parameter for the confidentiality and integrity mechanisms. It would have been possible to use separate values as the parameters to each mechanism, but in keeping with goal G1 and in trying to realise a lightweight approach, we have opted to use only one value. This does mean that if one of the mechanisms used is easier to break than the others, then by concentrating his/her efforts on that mechanism, an attacker could comprise all the mechanisms by breaking that one. While we have not conducted any qualitative analysis of the feasabilty of such an attack, we are confident that, given the known properties of the cryptographic functions that we have chosen, it would be extremely difficult for an attacker to succeed in an 'on the wire' attack of the system.

The shared-keys for an application are maintained in a local file that is read at start-up. The file is a table

that consists of two columns; the first column contains the DN of a peer, the next column the shared-key for that peer.

6.1 Authentication

The mutual authentication is realised by the use of signatures based on a keyed MD5 hash value. The key is K_{IR} the shared-key that is known only to I and R. The user application programming interface (API) to the authentication mechanisms is as follows:

The initiator token T_I is represented by the C structure:

```
typedef struct MLSInitiatorInfo_s {
    char *name;            /* string DN of user/application */
    char *peer;            /* string DN of peer user/application */
    unsigned int services; /* use values below */
#define MLS_pduIntegrity    (unsigned int) 0x00000001
#define MLS_confidentiality (unsigned int) 0x00000002
#define MLS_all             (unsigned int) 0x00000003
    char sessionKey[8];    /* MD5 key - zeros if no key for this assoc */
} MLSInitiatorInfo;
int makeMLSInitiatorAcseInfo(MLSInitiatorInfo *info, External **external);
int getMLSInitiatorAcseInfo(External *external, MLSInitiatorInfo **info);
```

The additional feature here that is not described in secion 5.2 is the `services` field; although this field was not mentioned previously it can be seen that it does not affect the earlier arguments regarding the mechanisms used. The field lets the initiator select the use of PDU integrity mechanisms and the PDU confidentiality mechanism. If the value is zero, then only the mutually authenticated connection is established.

Also, the timestamps are applied and checked beneath the API, the user being informed of any errors (see error list below), so there is no need for the user to handle the timestamp information.

Once the association is set-up, the session-key must be stored. The operation of the other mechanisms rely

on this. There is a simple key management API for session-keys and shared-keys (but this is not shown here). The APIs to the other mechanisms then use the file descriptor for the connection as a handle onto any security information for that connection.

The responder token, T_R is:

```
typedef struct MLSResponderInfo_s {
    char *peer;            /* peer */
    int error;             /* 0 (no error) or one of the #defines below */
    unsigned int services; /* use same values above */
} MLSResponderInfo;
#define MLS_miscellaneousError (-1)
#define MLS_unknownRecipient   (-2)
#define MLS_unknownInitiator   (-3)
#define MLS_credentialsExpired (-4)
#define MLS_sessionKeyReused   (-5)
#define MLS_invalidSignature   (-6)
int makeMLSResponderAcseInfo(MLSResponderInfo *info, External **external);
int getMLSResponderAcseInfo(External *external, MLSResponderInfo *info);
```

The responder may chose to ask for different services than those requested by the initiator. If the initiator does not agree with the `services` requested by the responder, then it can chose to close the connection.

In all the functions listed above, the error values returned are those listed above as `#define MLS_XXX` constants. `MLS_unknownRecipient` is a local API error as well as a remote error that can be returned by the responder. `MLS_unknownInitiator` is only a local API error to notify an error in the value given for `MLSInitiatorInfo.name`.

The `External` data structure is used to pass the information to the ACSE via CMISE; the `external` is transported in the CMIP `userInfo` which is in turn passed in to the ACSE `user-information` field.

The GoodBye token, G, is constructed directly, not requiring an intermediate C structure:

```
int makeMLSGoodByeAcseInfo(const int fd, char *name, External **external);
```

```
int getMLSGoodByeAcseInfo(const int fd, External *external, char **name);
```

`fd` is the file descriptor for the open connection which is to be closed.

The HASH function for signatures is realised by the MD5 implementation and is used as described in [34]. The signatures for the tokens are created using the following API:

```
int mlsSignature(PE pe, const char key[8], char signature[16]);
```

The `PE` is a **presentation element** abstraction used by ISODE. This API can be used to generate a signature for any syntax as long as an ISODE encoding (in the form of a `PE`) is constructed.

6.2 Confidentiality

The DES encryption and decryption functions are used in **cipher block chaining (CBC)** mode to realise the ENCRYPT and DECRYPT functions (respectively), hence the choice of 8 bytes for the size of the session-key, K_S, and the use of the initialisation value, V, also of size 8 bytes. The randomly generated value, V, not only forms the initialisation vector to the DES functions, but is also used for the sizes of $JP1$ and $JP2$:

$$\begin{aligned} |JP1| &= V[0] \bmod 128 \\ |JP2| &= V[1] \bmod 128 \end{aligned}$$

This precludes the need to include separate header information to the encrypted PDU, PDU_E, to give the location of the junk padding. The criteria for the choice of the sizes of padding value were fairly loose; the main considerations were:

- to generate enough padding to make a traffic analysis or known plain-text infeasible; the use of a random size of $JP1$ and $JP2$ upsets possible attacks based on knowledge of the PDU contents or knowledge of the byte alignment of the PDU encoding or size of the PDU.
- that the padding should not increase the final size of PDU_E so that it adversely affects the throughput with respect to the useful information contained within the PDU.

There could be an argument to change the padding size in light of the known performance of the MD5 algorithm implementation (explained later).

The API to this service is very simple and requires only the file descriptor of the open connection and the PDU to be encoded:

```
int encodeMLSPdu(const int msd, PE *pdu);
int decodeMLSPdu(const int msd, PE *pdu);
```

The file descriptor, `msd`, is used to look up the session-key using the key management API mentioned earlier.

6.3 Integrity

While the signatures for the PDU mechanism use the same HASH function as that used for the signatures for the tokens, the API is slightly different:

```
/* MD5 interface */
typedef struct MD5Value_s { /* 128 bits */
    union {
        unsigned char bytes[16];
        unsigned long words[4];  /* 32 bit long */
    } union_MD5Value;
} MD5Value;
int makeMd5Value(const int fd, PE pdu, MD5Value *check);
```

This takes account of the fact that the file descriptor, `fd`, is now available and can be used to access any security information related to a connection, e.g. the session-key. Other than that, the signature is formed in the same way as for the `mlsSignature()` API function.

The generation of the sequence numbers is by use of the standard C function `random(3)`. This has good randomness properties and the period of this function can be determined from its initial state. The first

4 bytes of K_S are used to generate the initiator to responder sequence, and the remaining 4 bytes to generate the responder to initiator sequence.

The application must record whether the connection was set up with itself as the initiator or responder:

```
typedef enum IRStatus_e {IRStatus_none, IRStatus_initiator, IRStatus_responder} IRStatus;
void setIRStatus(const int fd, const IRStatus s);
```

Once this is done, a simple API can be used to generate the sequence numbers:

```
int makeId(const int fd);
int makePeerId(const int fd);
```

An API also exists to save and restore the sequence state for a particular connection. This is needed in cases where the number sequence needs to rolled back when there are error conditions.

6.4 Performance

The speed of the various implementations that we have constructed are given here. They are based on tests performed on a Sun SPARCclassic with 48MB of memory running SunOS 4.1.3. Tests involving the network were performed using a Sun4 IPC with 24MB of memory also running SunOS 4.1.3. The two Sun workstations were connected by 10Mb/s Ethernet and were on the same spar of the network. The network was moderateley loaded. All mean values were calculated by timing 1000 repititions of the algorithms with buffer sizes of $2^n,\ n = 0, ..., 16$.

The DES implementation achieved a mean throughput of about 190KB/s (1.52Mb/s). The throughput of the MD5 algorithm implementation is non-linear at buffer sizes of less than 1024 bytes, but above this threshold value, it achieved a throughput of about 460KB/s (3.68Mb/s). Figure 1 shows this non-linearity.

If it is likely that the management systems will generally exchange PDUs that are smaller than 1024 bytes, using larger limits for the sizes of JPI and $JP2$ to force the algorithm towards operation nearer its greatest throughput *may* be desirable (available network capacity permitting, of course). However, in our case, the performance bottleneck is the DES implementation; its maximum throughput is approximately 190KB/s and the MD5 implementation achieves this rate at buffer sizes of ≥ 32 bytes.

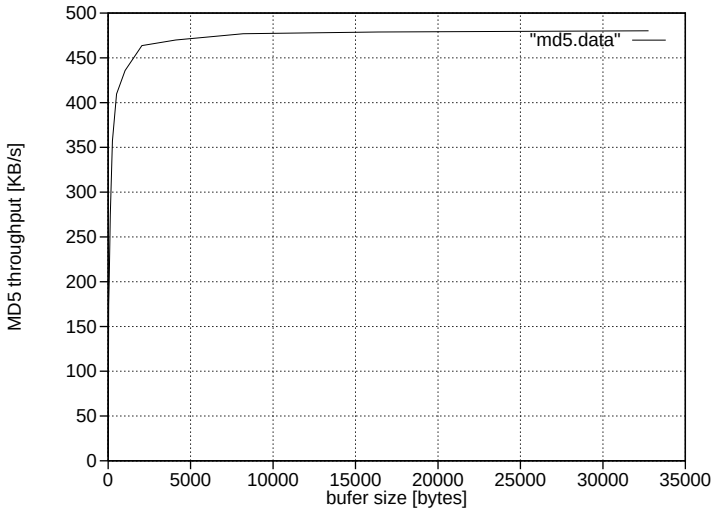


Figure 1: The performance of the MD5 implementation

In the X.509 public-key based approaches with we have we have experimented previously [4, 5], similar integrity check and confidentiality mechanisms were employed so the overhead is comparable in those respects. However, the connection set-up takes much longer, depending on the number of Directory operations required and number of certification authorities involved. We have previously recorded times between 11 and 24 seconds(!) with a fully public-key based approach, and these times did not include the checking of revocation lists that may have been held in the Directory. Although most of this extra time was due to the use of software implementations of the public-key cryptography functions, there was an overhead of at least 100% compared to connect times when there is no Directory access.

The figures in Table 1 shows the affect of the use of mutual authentication on typical connection set-up and tear-down times. While these times show that there is a relatively large percentage increase in the times due to the processing of the token exchange, the absolute times are still quite favourable.

(times in s)	**Authentication**		
	none	**mutual**	**overhead [%]**
connect	0.26	0.38	46
disconnect	0.04	0.07	75

Table 1: Management application connect and disconnect times

The overhead due to the integrity mechanisms was approximately 10% and that due to the confidentiality mechanism was approximately 12%. With both services together, the overall overhead was approximately 25% compared to the case when there were no security features in use.

Most of the overhead was due to the use of software implementations of the cryptographic algorithms used; the performance would improve significantly if hardware was used instead.

7 Discussion

The discussion here is presented as a list of issues that have arisen from our experiences with security of management work.

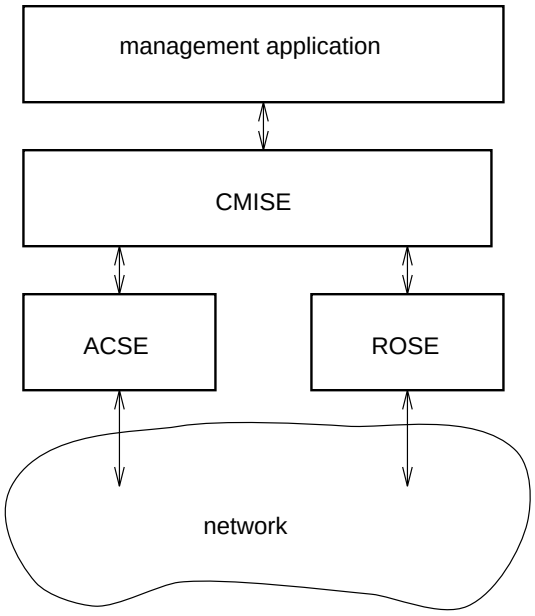
7.1 General applicability of results

The CMIP protocol is used by the **common management information service element (CMISE)**. The CMISE make use of two other services elements; the association control service element (ACSE) and the **remote operations service element (ROSE)** [31]. The interactions between the service elements and an application are depicted in Figure 2.

The CMISE uses the ACSE to establish and close connections. Once an association is established, CMIP PDUs are passed from the CMISE to ROSE and are carried as the payload of ROS PDUs. The mechanisms we have described pass credentials in the ACSE `user-information` field and evaluate signatures on ROS PDU payloads. We therefore suggest that it should be possible to apply these mechanisms to any ACSE/ROSE based protocol.

7.2 Key management: a brief comparison with a public-key based approach

The main drawbacks with the use of a shared-key system such as the one described here is that of managing the shared-keys for each I/R pair. The main problems are:



ACSE association control service element

CMISE common management information service element

ROSE remote operations servise element

Figure 2: The interactions between CMISE, ACSE and ROSE

- **Key distribution:** as the keys must be kept secret, they must be distributed in a secure manner if we are to have confidence in the security of this system. This is usually done out-of-band, for example, by secure e-mail, courier, etc. However, this problem can also affect public-key systems where the secret part of the key must be sent to the user.

  There is now a proposal for Kerberos, a shared-key based system, to use public-keys for the initial authentication exchange [35]. Such systems could be used to distribute shared-keys in-band, given a suitable key distribution framework.

- **Secure storage of keys:** the shared-keys must be stored securely so that only the applications/users that require them can read them. This problem is, again, also true of public-key based systems. However, hardware based solutions such as the use of SmartCards can solve this problem.

- **Revocation of keys:** should a key become compromised, it is up to the network administrators to ensure that the shared-key database is updated. The simple, lightweight system we have proposed does not have an associated authentication framework (such as in X.509) to allow the use of revo-

cation lists. This is in keeping with goal G2 to ensure that the management system is not reliant on another distributed service being available before it can operate.

- **Scaling:** a shared-key based solution does not scale well. For N parties, a shared-key based approach requires that the number of keys involved increase $\mathcal{O}(N(N-1)/2)$ whilst a public-key based approach scales $\mathcal{O}(N)$. This is a huge drawback and so we would favour the use of shared-key based systems in an environment where goals G1 and G2 are of paramount importance.

7.3 A question of trust

Establishment of trust in public-key based systems is based on the communicating parties verifying credentials that are signed by a trusted third party, the certification authority (CA). If the communicating parties do not share the same CA then there must be trust between their respective CAs. We say that the CAs have **cross-certified** each other, if some direct (possibly out-of-band) communication between the CAs has occurred to establish this trust. However, public-key systems can also set-up **certification paths** that are links of certificates between CAs that extend until a common point of trust between the communicating parties (i.e. a trusted CA) is reached. This sophisticated arrangement is required, ultimately, to establish trust between the communicating parties. Bringing the point of trust as close as possible to the communicating parties is a prime factor in achieving goals G1 and G2. The shared-key approach tries to reduce the complexity of the trust relationship to a minimum by assuming that trust exists directly between the communicating parties rather than being established via a trusted third party.

Public-key systems allow trust to be established between parties who may never before have communicated with each other. This is not possible with the kind of shared-key system described in this paper. However, in a well organised and well structured network, there is a high probability that network management system will consist of a known set of applications that are likeley to interact with each other in a well defined manner. In this case, the use of a secret-key based system to realise goals G1 and G2 may well have considerable advantages.

7.4 Hybrid architectures for security

Given the relative advantages and disadvantages of public-key versus shared-key systems, one must think carefully where they are to be used. One may consider the partitioning of the problem to take account of the existing structure of the network or system to be managed. For instance, we can consider the notion of a network **domain** as part of a network. A domain consists of network resources that can be logically grouped together in some way, for example, by all being the responsibility of a particular administrative body. We may look at the security requirements in terms of **inter-domain** communication and **intra-domain** communication, for example, in the general **telecommunications management network (TMN)** [36]. The information architecture for the TMN is depicted in Figure 3 showing interfaces between different functional blocks.

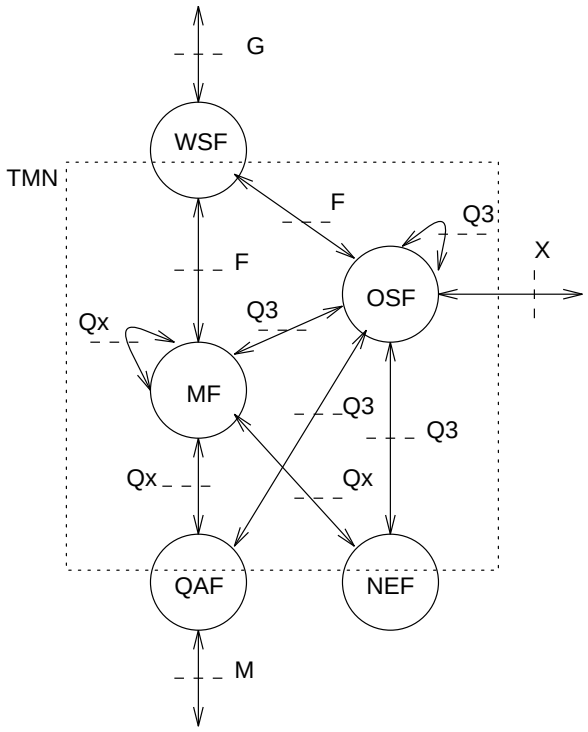
In this model we may consider that the X-interface is an inter-domain interface. As such, it may be likeley that the applications that communicate with this TMN at the X-interface will be external to the TMN and possibly unknown to it. Therefore, the use of public-key based security at this interface may be favoured.

However, the Q3-interface is an intra-domain interface, and at this interface we may have some confidence in at least the physical security of the network and so we can opt to use lightweight mechanisms. Here, we would gain performance advantages from using a shared-key based approach.

If we look at the G-interface and F-interface, we are faced with a slightly more complex problem. These two interfaces allow communication with entities outside the domain, but are bridged by a workstation function (WSF) that is on the domain boudary; it is likeley that the users of the WSF will be human administrators. At these interfaces we may decide to use either public-key or shared-key based approach, depending on the kind of applications or users we expect to use these interfaces.

The performance gains in using a shared-key system would be particularly significant when we consider that the TMN functional hierarchy depicts a layered architecture where many connection set-ups may be required between management applications at lower functional layers in response to a single management request submitted at a higher functional layer.

One thing that we have not considered in this paper is the existence of **security policy** within the system to control security related decisions. In our work, the decision functions are 'hard-wired' and the



TMN	telecommunications management network
WSF	workstation function
OSF	operations system function
MF	mediation function
QAF	Q adaptor function
NEF	network element function
- - -	interface to a TMN reference point

Figure 3: The TMN model

mechanisms have no real capabilities that allow us to tailor the use of the security features. For instance, at the G/F-interface, we may allow shared-key access during normal office hours when responsible network personnel are at hand, but only public-key access at other times when it would be abnormal for staff to be administering the network.

8 Summary and future work

We have designed, implemented and used a shared-key based security mechanism that provides authenticated associations, confidentiality of the PDUs and integrity checks for the PDUs. The mechanisms impelemented are relatively simple to use yet counter many kinds of security threats. In designing and implementing our system, our main goals were achieved, namely to produce a simple, lightweight, fast and self-contained system. The mechanisms are fairly general allowing the use of different algorithms; we have implemented them using the DES encryption standard and the MD5 hash algorithm. We have tested the performance of the system and found that the overheads incurred for the use of the security features are reasonable, and compares very favourably with previous work we have conducted that uses public-key methods. However, shared-key systems suffer from some major drawbacks, particularly in scaling, and so we believe they are best used in tightly coupled, well defined systems. The performance of the system could be improved greatly by the use of hardware implementations of the cryptographic algorithms used.

We have yet to investigate fully the use of particular security mechanisms in particular network environments, e.g. how best to integrate the use of both shared-key and public-key systems. We would also like to consider the use of security policies that can be interpreted and acted upon by the management applications and how these might affect the mechanisms in use. We are also currently working on access control mechanisms which could be used with the work presented in his paper to provide a complete security system for management systems. Indeed, access control policy would play an important role in defining TMN information model views, as the various interfaces in that model take into account management information visible at the interface to a particular functional block, and these views could be subject to constraints laid down in security policy.

References

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

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

[3] Network Management Forum, **Application Services: Security of Management**, OMNIPoint Network Management Forum 016, Bernardsville, NJ, August 1992.

[4] G. Knight, S. N. Bhatti, L. Deri, *Secure Remote Management in the ESPRIT MIDAS Project*, Proc. IFIP TC6/WG6.5 International Working Conference on Upper Layer Protocols, Architectures and Applications, Barcelona, Spain, 1 - 3 June 1994, [Elsevier Science B.V., Amsterdam, July 1994] pp 77-86

`http://www.cs.ucl.ac.uk/staff/S.Bhatti/papers/1994/ifip-ulpaa/paper.ps`

[5] S. N. Bhatti, G. Knight, D. Gurle, P. Rodier, *Secure Remote Management*, Proc. Fourth International Symposium on Integrated Network Management 1995, 1 - 5 May 1995, Santa Barbara, California, USA, editors A. S. Sethi, Y. Raynaud, F. Faure-Vincent, pp 156-169 [Chapman & Hall]

`http://www.cs.ucl.ac.uk/staff/S.Bhatti/papers/1995/isinm4/security/paper.ps`

[6] G. Knight, S. N. Bhatti, *Some experiences with secure management*, Proc. JENC6 - 6th Joint European Networking Conference, 15 - 18 May 1995, Tel Aviv, Isreal, editors J. Barbera & J. Kiers, pp 322/1-9.

`http://www.cs.ucl.ac.uk/staff/S.Bhatti/papers/1995/jenc6/paper.ps`

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

[8] P. R. Zimmermann, *The Official PGP User's Guide*, pp 216, [MIT Press 1995]

[9] C. Kaufman, *DASS – Distributed Authentication Security Service*, Internet RFC 1507, 10 September 1993.

[10] D. Maughan, B. Patrick, M. Schertler, *Internet Security Association and Key Management Protocol*, work in proress, IPSEC Working Group, draft-nsa-isakmp-01.ps, 6 July 1995.

[11] P. Karn, W. A. Simpson, *The Photuris Session Key Management Protocol*, work in progress, Internet Draft, Network Working Group, draft-ietf-ipsec-photuris-02.txt, July 1995.

[12] J. Galvin, K. McCloghrie, *Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC1446, 3 May 1993.

[13] J. D. Case, J. Galvin, K. McCloghrie, M. T. Rose, S. Waldbusser, *Security Protocols for Version 2 of the Simple Network Management Protocol (SNMPv2)*, work in progress, Internet Draft, draft-ietf-snmpv2-sec-ds-02.txt, 31 May 1995.

[14] K. McCloghrie, M. T. Rose, G. W. Waters, J. M. Galvin, *User-based Security Model for Version 2 of the Simple Network Management Protocol (SNMPv2)*, work in progress, Internet Draft, draft-kzm-snmpv2-sec-alt-00.txt, 30 June 1995.

[15] R. Atkinson, *Security Architecture for the Internet Protocol*, Internet RFC 1825, 9 August 1995.

[16] R. Atkinson, *IP Encapsulating Security Payload (ESP)*, Internet RFC 1827, 9 August 1995.

[17] R. Atkinson, *IP Authentication Header*, Internet RFC 1826, 9 August 1995.

[18] P. Metzger, P. Karn, W. A. Simpson, *The ESP DES-CBC Transform*, Internet RFC 1829, 9 August 1995.

[19] A. Ramanov, *Simple Authentication Mechanism for SNMP*, work in progress, draft-ramov-simple-snmp-00.txt, 27 June 1995.

[20] G. W. Waters, *Security Mechanisms for Version 1 of the Simple Network Management Protocol (SNMPv1)*, work in progress, Internet Draft, draft-waters-snmpv1-sec-mech-00.txt, 7 June 1995.

[21] A. I. Walten, *Security Encapsulation of SNMP*, work in progress, Internet Draft, draft-alten-snmp-sec-encap-00.txt, 30 July 1995.

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

[23] NIST FIPS Publication 180, *Secure Hash Standard*, National Institute of Standards and Technology, Federal Information Processing Standards Publication, U.S. Department of Commerce, 11 May 1993.

[24] NBS FIPS Publication 46-1, *Data Encryption Standard*, National Bureau of Standards, U.S. Department of Commerce, January 1988

[25] P. Rogaway, *Problems with Proposed IP Cryptography*, work in progress, Internet Draft, draft-rogaway-ipsec-comments-00.txt, 3 April 1995.

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

[27] ISO/IEC CD 10181-1, *Information Technology – Open Systems Interconnection – Security Frameworks in Open Systems – Part 1: Security Frameworks Overview*, 13 October 1992.

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

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

[30] D. Eastlake 3rd, S. Crocker, J. Schiller, *Randomness Recommendations for Security*, Internet RFC 1750, 29 December 1994.

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

[32] G. Pavlou, G. Knight, K. McCarthy, S. N. Bhatti, *The OSIMIS Platform: Making OSI Management Simple*, Proc. Fourth International Symposium on Integrated Network Management 1995, 1 - 5 May 1995, Santa Barbara, California, USA, editors A. S. Sethi, Y. Raynaud, F. Faure-Vincent, pp 480-493 [Chapman & Hall]

`http://www.cs.ucl.ac.uk/staff/S.Bhatti/papers/1995/isinm4/osimis/paper.ps`

[33] UCL Department of Computer Science, *The ISODE User's Manual*, Version 8.0, July 1991.

`ftp://bells.cs.ucl.ac.uk/src/isode-8.ps.Z`

[34] H. Krawczyk, *Keyed-MD5 for Message Authentication*, work in progress, Internet Draft, draft-krawczyk-keyed-md5-00.txt, 22 June 1995.

[35] C. Neuman, B. Tung, J. Wray, *Public Key Cryptography for Initial Authentication in Kerberos*, work in progress, Internet Draft, draft-ietf-cat-kerberos-pk-init-00.txt, 6 March 1995.

[36] ITU M.3010, *Principles for a Telecommunication Management Network*, Working Party IV, Report 28, December 1991.