A Secure Messaging Architecture Implementing the X.400-1988 Security Features

C. MITCHELL, D. RUSH* AND M. WALKER*
Department of Computer Science, RHBNC, London University, Egham Hill, Egham, Surrey TW20 0EX

The 1988 version of the X.400 Recommendations now include a number of security features designed to enable the provision of security services for Message Handling Systems. This paper describes one way in which the provided features can be used to provide a secure electronic mail system.

Received January 1989, revised March 1989

1. INTRODUCTION

1.1 X.400-1988 and security

The first version of the X.400 series of Recommendations, which specifies a store-and-forward message handling system, was adopted by the CCITT in 1984. During the 1984–1988 study period of the CCITT, these Recommendations have been completely revised, both to correct defects in the 1984 versions and to add new facilities. One notable example of new features is provided by the security elements present in the latest drafts of the 1988 Recommendations.[3,4,5,6]

These security elements can be used to provide a large variety of different security services, both end-to-end and connection-oriented. Many of the proposed security elements rely on key management services provided using security elements in the X.500 Directory Service Recommendations, and in particular on the service elements described in X.509.[8] Note that, unlike the X.400 Recommendations, the X.500 Recommendations did not exist prior to the current study period.

1.2 An X.400 based secure mail system

The purpose of this paper is to describe one way in which the security elements in the X.400 and X.500 Draft Recommendations can be used to provide a secure electronic mail system.

The security architecture described in this paper, although conformant to the CCITT Draft Recommendations, is just one of a number of possibilities. This is true for two main reasons. Firstly, there are a large number of possible security services that can be provided using the security elements in the Draft Recommendations, many of which will not be appropriate to all (or even most) mail systems. Secondly, in many cases, the same service can be provided in two or more different ways. It is certainly true that, for two "secure" X.400 systems to interwork, they will have to conform to a very detailed subset of the Recommendations, and it is hoped that this paper will contribute to the development of such "profiles".

* D. Rush and M. Walker are at Racal Research Ltd., Reading, England.

The security architecture described here has been developed as part of the Locator collaborative project, which is itself part of the Mobile Information Systems project, a major demonstrator within the UK Government sponsored Alvey programme. The partners within Locator are Hewlett–Packard Ltd, Racal Imaging Systems Ltd, Racal–Milgo Ltd, Racal Research Ltd and University College, University of London. Locator has as its goal the demonstration of a research prototype (X.400 conformant) secure messaging system with mobile access.

The prototype involves the use of three different types of message handling entity, namely User Agents (UAs), Message Stores (MSs) and Message Transfer Agents (MTAs). (Note that the reader is assumed to be familiar with the basic X.400 terminology.) In the mobile scenario, a mobile UA will communicate with a fixed MS/MTA, to retrieve and submit mail over a cellular radio link. Mobility makes the use of an MS essential, because a mobile UA will be unavailable for receiving mail for a large percentage of the time. These mobile UAs will be implemented on Hewlett–Packard 'Portable Plus' personal computers.

The mobile component in the main system had a significant effect on the design of the security architecture (as we see below) although the problems encountered and the lessons learnt are by no means confined to the mobile case. For example, many of the problems encountered will also have to be faced by designers of systems involving 'remote' UAs which are frequently disconnected from, and thus unable to receive mail from, the Message Transfer System.

Finally note that the demonstrator architecture and its physical implementation have been described in greater detail elsewhere.[9]

1.3 Contents of paper

The security architecture supports a number of security services which are described in Section 2. The provision of these services is considered in detail in Section 4, preceded by a discussion of key management issues in Section 3. The paper concludes, in Section 5, with a discussion of certain problems that have been encountered in using the security features in the X.400 Draft Recommendations.

2. SECURITY SERVICES

The security architecture has been designed to support a number of security services, all bar one of which are 'end-to-end' in nature. The selected services were identified during an initial study as being those most significant to end users of commercial messaging systems. The end-to-end services are: content confidentiality, message origin authentication, content integrity, non-repudiation of origin, replay detection and non-repudiation of delivery. The other service is access control which, within this architecture, is only provided on the UA-MS link, although Draft Recommendations X.411 and X.413,[5,6] allow it to be provided on all links between message handling entities. It should be observed that the service names we use are those in the X.400 Draft Recommendations; they do not correspond precisely to the names given in the OSI security architecture.[13]

Within the project's implementation of our architecture, not all the services may be requested independently. For instance, the message origin authentication, content integrity and non-repudiation of origin services are grouped together under a single user 'authentication service'. There are two reasons for this. First, the services may be provided using essentially one and the same mechanism, so that there are good practical reasons for grouping them. Second, it is considered unlikely that users of a mail system would want one of these services without the others, or indeed could really distinguish between them.

A second, and more significant, restriction on the provision of services is that an originator of a message may not simultaneously provide content confidentiality and request non-repudiation of delivery. The reason for this restriction stems from the fact that the Message Transfer System may need to deliver messages to an MS, rather than to the intended recipient UA. This rather undesirable restriction on the provision of security services in the mobile environment is discussed in detail in Section 5.2. It serves to illustrate the problems that arise when attempting to use the X.400 security features to provide a comprehensive set of security services, and to pin-point where amendments to the standards are needed.

3. KEY MANAGEMENT

In our architecture, the management of the cryptographic keys, required for the provision of the selected security services, is achieved by using security features built into the Directory Service specified in the X.500 series of Draft Recommendations. Of particular importance is the authentication framework, specified in Draft Recommendation X.509.[8]

The key management system is based on the use of public key (asymmetric) cryptosystems (pkcs), which are used for both digital signatures and encryption. In a pkc, keys are produced in pairs, one of which is made public whilst the other is known only to its owner.[2] The X.509 authentication framework allows a user's public key to be stored in its directory entry. One user wishing to exchange secure messages with another obtains the other user's public key from the appropriate directory entry, and then uses this key to provide the required security services, as described in Section 4.

3.1 Digital Signatures

The X.509 authentication framework does not specify any particular pkc, although it does require the use of a pkc satisfying a special property. This property (listed in clause 6.1 of X.509,[8]) is that 'both keys in the key pair can be used for encipherment', i.e., both the public key and the secret key can be used to operate on arbitrary data. The reason for insisting on this special property is that the framework specifies how the pkc is to be used to provide digital signatures, and this method requires that the pkc has the specified property.

The method specified for digital signatures is simple. First, the data to be signed is 'hashed' using a collision-free hash function.[10] In our architecture the hash function used is that suggested in Annex D of X.509, and is based on the repeated use of modular squaring. The end result of this hash function (the 'hash value') must be sufficiently small to be processed by a single encryption operation of the selected pkc. The hash value is then encrypted using the selected pkc under the control of the secret key of the signer. Since the hash function is public, the signature can then be checked by anyone who has access to the public key of the signer. As in X.509,[8] we denote the signing of data block I using the secret key of signer X by

$$X\{I\}$$

where $X\{I\}$ is defined to consist of a copy of I followed by the value obtained from hashing and enciphering I under the signer's secret key.

Unfortunately, there are very few pkcs known which have the specified property, and the only well-established one is RSA;[18] consequently, this is the pkc we use in our demonstrator. There is no need to specify so precisely how digital signatures are produced, and the fact that this is done within X.509 is a shortcoming of the authentication framework. Indeed, a small change to the framework would allow arbitrary pkcs to be used with arbitrary (and possibly unrelated) digital signature algorithms.

3.2 Certificates and Certification Authorities

Since the directory is not a secure or trusted service, means need to be provided for users to verify public keys read from the directory. As described in X.509,[8] this is achieved by using off-line trusted entities known as Certification Authorities (CAs) who provide 'certificates' for users' public keys.

In order to store a copy of a public key in the directory, a user must choose a CA; this CA must be trusted by the user, because a fraudulent CA has the power to mislead the users for whom it acts. Like the users, every CA must also have its own pkc pair.

The user and the CA exchange their public keys in such a way that each trusts the validity of the received key and the identity of the other party; this could, for example, be achieved by the user and the operator of the CA meeting and physically exchanging floppy disks containing the relevant keys. The CA then computes a digital signature on the following set of data: the CA's name, the user's name, the user's public key and the period of validity for the user's public key. This signature is computed with the CA's secret key, using the technique described above. The set of data, together

with the signature itself, forms the user certificate, which is stored in the user's directory entry. Using the notation introduced earlier, if user named A has certification authority with name X, then A's certificate has the form

$$X\{X, A, Ap, T^A\}$$

where Ap is A's public key, and T^A indicates the period of validity of the certificate. We denote such a certificate by

$$X\langle\langle A\rangle\rangle.$$

Any other user, which has a trusted copy of X's public key, can then check the signature on the certificate, and hence obtain a verified copy of A's public key.

The scheme, as described so far, no longer works when two users are served by different CAs. To cover this possibility, CAs may generate certificates for each other; such certificates are called 'cross-certificates'.

As an example of the use of cross-certificates, suppose users A and B are served by CAs X and Y respectively. Then X and Y generate (and store in the directory) the following certificates:

$$X\langle\langle A\rangle\rangle, Y\langle\langle B\rangle\rangle.$$

Additionally suppose that CAs X and Y are able to exchange public keys in a verifiable way, and thence generate the following two cross-certificates:

$$X\langle\langle Y\rangle\rangle, Y\langle\langle X\rangle\rangle.$$

Then user B may use the sequence of certificates

$$Y\langle\langle X\rangle\rangle, X\langle\langle A\rangle\rangle$$

(in combination with a trusted copy of Y's public key) to first obtain a verified copy of X's public key, and then a verified copy of A's public key. Such a sequence of certificates is called a certification path. It is important to note that, in order to trust a public key checked using such a path, it is necessary to trust all the CAs in the path.

In our prototype system we have not implemented any sophisticated mechanisms for constructing certification paths. Indeed the demonstration system is sufficiently small to allow each CA to cross-certify each other CA, and therefore certification paths will only ever contain two certificates. However, in larger systems, it will be necessary to provide for the construction of longer paths, and in general this could present a very difficult problem.

4. SECURITY SERVICE PROVISION

The majority of the security features incorporated into the X.400-1988 Draft Recommendations make use of a cryptographic construct called a token. In fact, tokens will be used in the provision of all the security services in the project demonstrator. We begin this section by describing the general form of a token, and then consider how such a construct is used in the provision of the various security services.

4.1 Tokens

A token consists of a series of data fields with a digital signature appended, exactly like a certificate. Unlike certificates, tokens are always generated by a user for transmission to a single other user. The precise form of a token sent by user B to user A is

$$B\{t^B, A, \text{sgnData}, Ap[\text{encData}]\}$$

where t^B is a timestamp, A is the name of the intended recipient and sgnData and encData are collections of security-related parameters; the contents of sgnData and encData vary depending on the security services being provided. The notation Ap[encData] means that the data field encData is sent encrypted under A's public key, so that the contents of encData are available only to the intended recipient. In our architecture, encData is only ever used for the transmission of secret key information as part of the content confidentiality procedure described below. It is important to note that, whatever the contents of the sgnData and encData fields, the signature on the token prevents them being changed in an undetectable way.

It should be observed that the token signature is applied after the encData has been encrypted under the recipient's public key. This is the construction that is specified in X.411 and X.509,[5,8] but it is arguable that encryption should be applied after the signature, either to the entire token, or to selected 'secret' fields of the token. In many cases the chosen token structure could lead to security problems, although these are overcome within our architecture by a suitable choice of hashing function; for further discussion see Section 5.1 below.

Within our architecture, all the end-to-end services are provided using a special type of token called a message-token. Some of these services may be provided in other ways, but we concentrate here only on those techniques we use. Message-tokens accompany individual messages, on a per-recipient basis, i.e., a distinct token is sent to each recipient of the message for whom security services are provided.

4.2 Content confidentiality

If content confidentiality is required for a particular message, then, unlike other security services, this must be provided either to all, or to none of the recipients. The service is provided by encrypting the entire message content. The encryption algorithm used for this purpose is not specified in the X.400 Recommendations, and may be either conventional (symmetric) or public key (asymmetric) in type. In our architecture the DES algorithm is used.[1,12] The key used for the encryption process is selected at random by the message originator, a new key being selected every time a confidential message is sent. A different message-token is sent with the message for every recipient, and the key used to encrypt the message content is included in the encData field of each token.

4.3 Authentication services

In the demonstrator architecture, the three authentication services, message origin authentication, content integrity and non-repudiation of origin, are all provided together, and may not be requested independently. In fact, the mere existence of a message-token for a recipient provides message origin authentication for that recipient, although the service is of dubious value on its own since there is no guarantee

that the message content has not been altered. This is one reason why all three services are combined, resulting in a meaningful and useful set of options being offered to users.

To provide these three services a 'Content Integrity Check' (CIC) is generated and included in the sgnData of the message-tokens for all recipients for whom the services are to be provided. This CIC must be computed as a 'one-way function' of the message-content. The function to be used to compute the CIC is not specified within Draft Recommendation X.411.[5] However, if it is to provide the non-repudiation of origin service, then it must satisfy the same properties as are required of a hash function used in the computation of digital signatures. In our architecture the CIC is computed using precisely the same function as that used for hashing data in certificates and tokens, i.e., the modular-squaring function described in Annex D of Draft Recommendation X.509.[8] The presence of the CIC in a token enables a recipient to verify the integrity and authenticity of the message-content.

It is interesting to note that the relevant Draft Recommendation, i.e. X.411,[5] allows the CIC to be sent independently of the token. As discussed in Ref. 16, in this case it must be computed using a secret key known only to the originator and recipient; otherwise it is unprotected against manipulation by an active interceptor.

Even if the CIC is computed using a secret key, problems still arise when messages are to be sent to more than one recipient. As described in Refs 14 and 15, depending on the method used to compute the CIC, it may be possible for one recipient to modify the version of the message intended for another recipient in an undetectable way. Since there is no means for providing a different CIC for each recipient, one of the remedies described in Ref. 15, for avoiding this problem may be rather difficult. These problems appear to make the inclusion of the CIC in the token the preferred mode of operation in most circumstances.

The replay detection service is provided to a recipient by including a message sequence number within the sgnData of the message-token for that recipient. This service forms part of the message sequence integrity service described in Draft Recommendations X.400 and X.402.[3,4] The way in which the sequence number is used is not completely specified within the X.400 Recommendations; we now describe how it is used in our architecture. Every user keeps a list of all other users for whom this service is to be provided, and a number is associated with each entry in the list. This number represents the message sequence number assigned to the last message sent to that user. The next time a message is sent to that user it is assigned a sequence number one larger than the stored value, and the stored value is updated. The receiver of this message will also keep a list of numbers (one for each other user from whom messages are received that incorporate replay detection). This enables the message sequence number in the message token to be checked for its 'freshness'. The inclusion of the sequence number in the token ensures its integrity, and therefore provides a secure replay detection service.

In general, given that a sequence number approach is followed, means must be provided for dealing with the exhaustion of the message numbering space. One possibility, and the approach followed within our architecture, is to make the space sufficiently large so that the problem will never be encountered in practice. Another possibility is to allow sequence numbers to cycle, i.e. to go from the largest allowed value back to zero. In this case a window criterion must be added, i.e. for some small pre-determined value k, a sequence number is only accepted if it is at most k further on in the numbering sequence than the number of the last acceptable message.

4.4 Non-repudiation of delivery

The final end-to-end service, namely non-repudiation of delivery, is rather different in nature from the other services, in that it is provided in two stages: request and provision. The originator of a message does not provide the service, but rather requests its provision. The service is actually provided by the message recipient through the return of a 'receipt' for the message called a 'proof-of-delivery'.

The procedure for requesting the service is to include a 'proof-of-delivery-request' flag in the message token for the recipient(s) concerned. When such a message is delivered, the proof-of-delivery is computed as a digital signature on the (unencrypted) message-content and various delivery-related parameters. The signature is evaluated using the recipient's secret key, and the function used is precisely the same as that used for signatures on certificates and tokens. The proof-of-delivery is then returned to the message originator within the delivery report, and can be used by the originator to give the desired non-repudiation of delivery service.

4.5 Secure access control

We conclude this discussion of security services by describing how the only security service which is not end-to-end, namely secure access control, is provided. As has already been mentioned, within the demonstrator architecture this service is only provided on the link between a UA and its associated MS; this is because the UA-MS link is perceived as being the most prone to attack. The service is based on the exchange of another special type of token, called a bind-token, at the time a connection is set up between a UA and its MS. The service is restricted to access control, and does not provide connection integrity or confidentiality. However, the bind-tokens could be used to exchange keys for the provision of such connection-oriented services, although no means of providing them are defined within the X.400 Draft Recommendations.

In more detail, the initiator of the connection between a UA and its MS, which must be the UA, includes a bind-token in its initial communication; for details of the connection initiation see Draft Recommendation X.413.[6] Prior to generating the token, the connecting UA selects a random number specifically for this connection. The size and form of the random number is not specified within Draft Recommendation X.413; however, we use a 64-bit value. This number is then included in the sgnData field of a bind-token sent from the UA to its MS; this token is called an 'initiator-bind-token'. The encData field of the token will be empty,

although, as mentioned earlier, this could be used to convey keys for providing connection-based security services.

On receipt of an initiator-bind-token, the MS checks the signature on the token and recovers the random number from the sgnData field. The MS also checks the time value within the token in order to check that it is 'fresh'. Given that the received token is deemed acceptable, the MS generates another bind-token, called a 'responder-bind-token', and returns it to its UA. The random number taken from the sgnData field of the initiator-bind-token is reproduced in the sgnData field of the responder-bind-token. On receipt of the responder-bind-token, the originating UA checks the token signature and the random number in the sgnData field, and if these tests pass, the connection is allowed to proceed.

5. PROBLEMS WITH USE OF THE X.400 MECHANISMS

We now consider some problems that have been encountered with using the security features within the X.400 Draft Recommendations to provide the required security services in the mobile environment. The major problems have centred around the structure of tokens and the provision of the non-repudiation of delivery service. We now consider these problems in a little more detail.

5.1 Token structure

As we noted in section 4.1 above, and as has been pointed out by Burrows and Needham,[17] there are considerable potential security problems with the form of token used in the X.400 and X.500 Recommendations. The root case of these problems is that the X.509 token involves signing encrypted data, which is generally accepted as bad practice; see, for example, Section 9.3 of Davies and Price.[11] This is because, if the crypto-operations are performed in that order, the recipient will only be able to guarantee the authenticity of the encrypted data, and not of the plaintext data. This leads us to the main objection to the X.509 form of token.

To explain this objection we need to first consider a 'new' security service, which, although not normally part of an authentication/integrity service, is intimately related to them. For want of a better name we call the service 'Authorship'. This service, if provided for a message sent from user A to user B, will guarantee to B that the message is known to A (the service is clearly not useful unless provided in conjunction with integrity, authentication and confidentiality services). It is reasonable to ask in what circumstances such a service would not be provided by default, given the provision of all these other services; we return to this question in a moment. We first consider what use might be made of such a service.

As an example consider its potential usefulness to Patent Offices of the future, to which claims are submitted via electronic mail (or some similar electronic communications medium). Individuals submitting patent claims to the office would, presumably, want to provide confidentiality, integrity and authentication services for the patent claim message. The recipient of a claim would also, more than anything else, want to ensure that the patent claim comes from whom it claims to come from, since he will ultimately be assigning ownership of the patent to the claimed originator.

We now consider how such a service can fail to be provided. Suppose that, as in our architecture,

—Message integrity is provided by including a Content Integrity Check (computed as a function of the message content) in the sgnData field of the token;
—Originator authentication is provided by the signature on the token;
—Message confidentiality is provided by using a randomly chosen key to encrypt the message content, and this key is passed to the recipient in the encData field of the token.

Now suppose that a malicious third party, wishing to claim 'authorship' of the message, intercepts it in transit. He now constructs a new message, containing the same encrypted content as the old message, but accompanied by a new token. This new message is then sent to the original intended recipient by the third party. The new token contains the same encrypted encData field as the old token, and the same Content Integrity Check in the sgnData field, although the token is now signed by the third party. The recipient of this new message will now believe that the message comes, not from its true originator, but from the third party. Therefore the 'authorship' service is not provided, although message confidentiality, integrity and origin authentication are!

It appears reasonable to lay most of the blame for the lack of provision of the authorship service with the form of token used. If the encryption and signature operations were performed in reverse order, the service would be provided 'for free'. This is because, if the signature is always computed on unencrypted data, the recipient can establish that the claimed originator knows the plaintext values of all the data in the token, and therefore the plaintext value of the message.

One might argue that the inclusion of the name of the originator in the message itself would solve the problem, since any changes to the message would be detected by the integrity check mechanisms; indeed, the name of the originator may (optionally) be present in the P2 encoding of a message, as defined in X.420.[7] However, this, like other possible solutions, is very much a 'fix' to a problem which would not exist if the token did not possess a defect in the first place.

Having made these points we note that, in our architecture, the problem is not present because of the method used to compute the CIC. As stated above, the CIC is computed using the hash function described in Annex D of X.509,[8] which is based on the use of repeated modular squaring. The modulus used is always the modulus for the RSA key of the signer, and the CIC is therefore a function of the identity of the message originator. This prevents its misappropriation by third parties wishing to claim ownership of messages.

5.2 Non-repudiation of delivery

In the description of the generation of the proof-of-delivery by a message recipient, the precise identity of the key used to compute the signature was deliberately not discussed, and was merely referred to as the recip-

ient's secret key. Problems arise because of the fact that Draft Recommendation X.411,[5] requires that the proof-of-delivery be generated and returned to the delivering MTA at the time the message is delivered. If the message is delivered to an MS, then the MS must generate the proof-of-delivery rather than the intended recipient UA. This requires the MS to have access to an RSA key pair.

One solution to this problem would be to give every MS access to the secret RSA key belonging to its associated UA. This solution has a major drawback in that many MSs may be implemented on the same remote machine, which is not trusted to the same degree as the portable UA machine. The only other solution, and the solution adopted for the demonstrator architecture, is to equip every MS with its own RSA key pair, distinct from the key pair belonging to its associated UA. The secret key from this MS key pair is then also used to sign the responder-bind-token used in the provision of secure UA-MS access control.

With this solution, problems arise with generating certificates for the MS public keys. Such a certificate must be distinguishable from a UA certificate, or else possessors of MS keys will be able to masquerade as UAs. Within the current draft of the X.509 authentication framework,[8] there is no provision for doing this, since a UA and its corresponding MS share the same O/R name. In our work we have therefore been forced to use non-standard means to achieve our aims.

One possibility under consideration is to commandeer a field of the certificate and use it to distinguish MS and UA keys. In addition to names, dates and a public key value, every certificate contains an 'algorithm-identifier' (within the public key parameter), intended to specify which algorithm the public key is intended to be used with (e.g. RSA). In our architecture we are considering the use of this data field to include additional information indicating the 'scope of use' of the key, i.e. whether the key belongs to a UA, an MS or to a CA. This is admissible since there are, as yet, no standards for algorithm-identifiers; however, the scheme does violate the spirit of the ISO naming and addressing conventions. This is the only part of the architecture in which we are considering using non-standard security facilities.

A further problem arises when an MS is required to compute a proof-of-delivery. We stated above that the proof-of-delivery is computed using the unencrypted message-content. This is a problem for an MS, since the key required to decrypt the message-content is within the encData field of the message token, encrypted using the recipient UA's public key. It is therefore not possible for the MS to recover the unencrypted message-content unless it has access to the UA's secret RSA key. As has already been stated, this is undesirable, and not allowed in our architecture. The 'solution' we have adopted is to prohibit an MS from providing non-repudiation of delivery if the message content is encrypted. This is clearly unsatisfactory, but there is no obvious way to improve the situation.

5.3 Concluding remarks

In conclusion, we have identified two significant problems with the current versions of the X.400 and X.500 Draft Recommendations. Although these problems are not catastrophic, in that it is still possible to build a secure electronic mail service, it is nevertheless of vital importance that they be addressed within the next study period of the CCITT.

6. ACKNOWLEDGEMENTS

The work described in this paper was carried out as part of the UK DTI Alvey Programme in the Mobile Information Systems Demonstrator. The authors would like to acknowledge the help and support of their colleagues on the Locator project who have collaborated in the design of the security architecture described in this paper.

REFERENCES

1. A.N.S.I. X3.92-1981, *Data encryption algorithm*, American National Standards Institute, 1981.
2. H. J. Beker and F. C. Piper, *Cipher systems*, van Nostrand, U.K., 1982.
3. C.C.I.T.T., *Draft Recommendation X.400. Message Handling: System and Service Overview*, Version 5.5, April 1988.
4. C.C.I.T.T., *Draft Recommendation X.402. Message Handling Systems: Overall Architecture*, Version 6, Geneva, March 1988.
5. C.C.I.T.T., *Draft Recommendation X.411. Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures*, Version 6, Geneva, March 1988.
6. C.C.I.T.T., *Draft Recommendation X.413. Message Handling Systems: Message Store: Abstract Service Definition*, Version 6, Geneva, March 1988.
7. C.C.I.T.T., *Draft Recommendation X.420. Message Handling Systems: Interpersonal Messaging System*, Version 6, Geneva, March 1988.
8. C.C.I.T.T., *Draft Recommendation X.509. The Directory—Authentication Framework*, Geneva, March 1988.
9. R. Cole, C. Hall, M. Hassall, A. Pell and J. Walker, Demonstrating the mobile office, *Proceedings of UK IT 88, Swansea, July 1988*, IEE, 1988, pp. 597–600.
10. I. B. Damgard, Collision free hash functions and public key signature schemes, *Proceedings of Eurocrypt 87*, Springer-Verlag, Berlin, 1988, pp. 203–216.
11. D. W. Davies and W. L. Price, *Security for computer networks*, John Wiley and Sons, Chichester, 1984.
12. F.I.P.S. 46, *Data encryption standard*, National Bureau of Standards, 1977.
13. I.S.O., *IS 7498-2. Information processing systems—Open Systems Interconnection—Basic Reference Model. Part 2: Security Architecture*, International Organisation for Standardisation, 1988.
14. C. J. Mitchell, Multi-destination secure electronic mail, *The Computer Journal* **32** (1), 13–15 (1989).
15. C. J. Mitchell and M. Walker, Solutions to the multi-destination secure electronic mail problem, *Computers and Security* **7** (5), 483–488 (1988).
16. C. J. Mitchell, P. D. C. Rush and M. Walker, CCITT/ISO standards for secure message handling, *IEEE Journal on Selected Areas in Communications*, to appear.
17. R. M. Needham, Private communication, 1988.
18. R. L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21**, 120–126 (1978).