Key Management I

Designing Secure Key Exchange Protocols*

Colin Boyd and Wenbo Mao

Communications Research Group,
Electrical Engineering Laboratories,
University of Manchester,
Manchester M13 9PL, UK
Email: Colin.Boyd@man.ac.uk

Abstract. Protocols for authentication and key exchange have proved difficult to develop correctly despite their apparent simplicity in terms of the length and number of messages involved. A number of formal techniques have been developed to help analyse such protocols and have been useful in detecting errors. Nevertheless it is still difficult to be certain that a particular protocol is correct.
This paper explores a different approach; instead of analysing existing protocols the aim is to design protocols to be secure in the first place. A methodology is developed for designing key exchange protocols in a restricted way such that they must be correct according to a defined security criterion. The protocols are defined abstractly with the cryptographic operations specified only according to their basic functions. This allows the protocols to be made concrete in a variety of ways. A number of concrete protocols are presented, some of which appear novel and, at the same time, efficient in comparison with existing ones.
Keywords: Cryptographic protocols, key management, authentication.

1 Introduction

The difficulty of correctly designing secure cryptographic protocols for authentication and key exchange has become widely recognised in recent years. As a result a number of formal techniques have been developed to enable analysis of such protocols [3, 7, 22, 13, 11]. These methods have proved successful at finding problems with existing protocols, sometimes previously unrecognised ones. Unquestionably formal analysis techniques have helped to further understanding in how protocols work, or fail to work, by providing a language in which to mathematically describe and argue about them. Despite these successes, there remains a great deal of doubt as to whether any of the existing techniques is sufficient to provide a proof that a given protocol is sound [15, 23, 2, 8]. The situation seems to have a fair analogy in testing for computer programs; rigorous testing allows many bugs to be found but will not provide a proof of correctness.

* This work is funded by the UK Science and Engineering Research Council under research grant GR/G19787.

The experience gained in correct design of computer programs provides a lesson for correct protocol design. It is now well understood that to develop a correct program the best method is to design it formally in an abstract way and then to proceed towards a concrete implementation [10]. The difficulty of proving that an existing program is correct is usually far greater. In this light it seems obvious that we should develop techniques to design protocols that are guaranteed to be correct in the first place, rather than design them with *ad hoc* techniques and try to remove any bugs by formal analysis afterwards [8].

This paper describes a technique to design key exchange protocols which are guaranteed to be correct in the sense that a specified security criterion will not be violated if protocol principals act correctly. The technique is developed from basic cryptographic properties that can be expected to be held by a variety of cryptographic algorithms. Protocols can be developed abstractly and any particular type of algorithm that possesses the required property can then be used in a concrete implementation. Building on previous work [1] protocols may be classified according to what secure channels initially exist.

The idea of the technique is to restrict attention to protocol messages which contain a small number of elements which have a well defined purpose and meaning. This implies that only a restricted set of protocols can emerge as a product of the technique. Nevertheless, comparison with existing protocols indicates that the protocols developed are both efficient and flexible enough to accommodate most reasonable requirements.

The remainder of the paper is organised as follows. In the next section the model of security is explained using some mathematical formalism. The security criterion appropriate for this model is defined and it is shown that if principals act according to a set of simple security assumptions then the security criterion will not be violated. These assumptions are then used as guidance to develop the basic format of messages. Generic protocols with this message format are suggested which can be made concrete in a particular design. Section 3 includes protocols suitable for common situations designed using the technique. These include user-to-user protocols, protocols using a trusted third party and conference protocols. Concrete implementations of the protocols are suggested and compared with existing protocols. The final section examines limitations of the technique and possible further developments.

2 The Model of Security

In this paper the only issue to be addressed is that of secure key exchange. Protocols aimed solely at authentication are not addressed. There are at least two reasons for this. One is in order to concentrate on a single achievable goal. Another is that there is some difficulty in agreeing exactly what is meant by authentication in a general sense. Nonetheless authentication plays a crucial part in key exchange protocols, but here it is clearer exactly what is required. It will be assumed that a successful run of a protocol should achieve the following goals for a participating user.

1. The user should possess a new key for use with a set of users $\mathcal{U}$.
2. The new key should not be known by any non-trusted user except those in $\mathcal{U}$.

It will be assumed that the new key is a shared symmetric key for use in a single session, but existing keys used during the protocol may be symmetric or asymmetric. The above goals are uncontroversial but it is frequently desired to include further goals. For example it may be desired that the user should know that all the users in $\mathcal{U}$ are in possession of the new key. This can be achieved by further mechanisms if desired and may properly be seen as distinct from the mechanism to enable key exchange. Note that $\mathcal{U}$ will often be a single user but there is no reason to make this distinction in the model and not doing so allows the inclusion of conference key protocols.

A large variety of key exchange protocols exist. It is not at all obvious why there are so many nor what the differences are between them. Some of the issues that may be important in the design of a particular protocol are as follows.

- A specific set of goals may be desired for the protocol. These will always include those expressed above. Others may be achieved subsequently. The protocols designed in this paper may be extended to achieve further goals.
- There may be constraints on the order in which messages are sent and the channels available to send them on. For various practical reasons communications channels may not exist between every pair of principals and there may be reasons why one principal or another must initiate communications. In the protocols of this paper the physical path a message takes is of no importance. The ordering of messages is also irrelevant except that in some cases certain messages cannot be formed until others are received.
- Different principals may have different computational capabilities. In particular only some principals may be competent to generate good keys. The type of cryptographic algorithm available, particularly public key algorithms, may also be restricted. The two most common situations are where a user generates a key intended for use with one other user, and where a trusted party generates a key for one or more users. These are the two cases covered in this paper. Variations allow any type of cryptographic algorithm to be used as long as the basic requirements for secure communications [1] are satisfied.

Despite the flexibility of the protocols proposed below, they appear economical in comparison with most existing key exchange protocols. This results from deciding exactly which message components are required and which of these should be processed cryptographically. If it is felt desirable to avoid special cryptographic opportunities for attack, such as *known plaintext attacks* [4], further processing may be used.

2.1 Cryptographic keys and secure channels

The secure channels available in a particular architecture can be described in terms of the cryptographic keys known to the participants. This information

alone is sufficient to decide whether it is possible to arrange for secure communications between every pair of users [1]. The model described here uses a similar idea; it is mathematically very simple relying on just a few sets and functions between them. Two fundamental sets are assumed whose structure is outside the concern of the model. The set *User* consists of principals in the system who will participate in protocols; it is of no concern to the model how different users are named. The set *Key* contains all keys that are available for use with the cryptosystems of interest in the system. The length and value of any key is of no concern. The cryptographic information stored by each user is modelled as a set of ordered pairs connecting a key and a user[2].

$$LocalSecret : User \rightarrow (Key \leftrightarrow User)$$

For example if user A associates key k with user B then this is modelled as

$$(k, B) \in LocalSecret(A)$$

The set *Key* can be thought of as partitioned into three subsets *Private*, *Public* and *Shared*. These accommodate the difference between conventional symmetric (or shared-key) cryptography and asymmetric (or public-key) cryptography. The *dual* is defined for every key; for a shared key the dual is the key itself but for a public key the dual is the corresponding private key and *vice versa*. A user who associates a shared key with one or more other users must know the value of that key. However a user who associates a private key with a user will know only the dual public key.

Consider now what it means for the system to be secure from the viewpoint of an individual user *Alice*. There are two operations that Alice may perform using the keys that she knows about.

1. Alice may use a key to send a confidential message to another user. This will be achieved by encrypting the message using a cryptographic algorithm which allows only those users in possession of the decrypting key to recover the message. In the case of a shared key it is necessary that all those users who possess this key are known to Alice. It does not matter, from the security viewpoint, if some of the users in Alice's list do *not* in reality possess the key. In the case of a public key it is necessary that all those users who possess the dual private key are known to Alice. As long as this is the case Alice will not be surprised by who gets her message.
2. Alice may use a key to authenticate a message received by her. It can be verified that in this case also she will not be surprised by who sent the message as long the same condition holds. That is, Alice must know all users who possess the private, or shared, key required to form the authenticated message.

In order to model this security condition a global notion of which users may be in possession of which keys is required. This is a set of ordered pairs as follows.

[2] The notation $A \leftrightarrow B$ denotes the set of relations between the set A and the set B

$$GlobalSecret : Key \leftrightarrow User$$

Keys are issued by authorised principals and we will assume, for simplicity that there is a single principal, called the *server* who does this. The set *GlobalSecret* does not conform to any real system variable but is an abstract modelling device. The set records all the keys k which have been issued by the server and which are good, in the sense that k is known to user U implies $(k, U) \in GlobalSecret$. In a practical interpretation, the *GlobalSecret* set includes all key encrypting keys shared between the server and each user, and all session keys which have not expired. Of course no user is able to decide with certainty which keys are good (and so in the *GlobalSecret* set) and which are not. In practice each user will have to believe that newly generated keys received from the server are still good, and hence will be in the set. For the moment we will ignore this issue.

The above notion of security may now be defined in terms of these sets. For a particular key k used by Alice there will be a set of users in her set *LocalSecret* (that is $LocalSecret(Alice)$) which she associates with that k (or its dual if it is a public key). All users U with (k, U) in *GlobalSecret* should be contained within this set known to Alice. This can be defined in one equation as follows.

Security Criterion (for Alice)

$$\forall k \in \mathrm{dom}(LocalSecret(Alice)) \bullet GlobalSecret(k) \subseteq LocalSecret(Alice)(k)$$

All this says is that for those keys which Alice has in her list, she has all those names against it which are in the *GlobalSecret* set. Note that it is an implicit assumption in this equation that the only keys in Alice's list are also in the global list; in other words, only good keys are in her list.

The method of protocol design in this paper is to only allow messages which leave this condition true. The server always generates and sends a *new* key which therefore (formally) does not exist in the *LocalSecret* of any user. A user's *LocalSecret* set is only updated when a key is received from the server. Messages contain information on who else has been sent the key and hence who is linked with that key in *GlobalSecret*. Users ensure that keys received from the server are new and can therefore be assumed to exist in the global set. We may summarise the assumptions required to maintain security as follows.

Security Assumptions

1. The trusted server generates a new key k each time it is activated.
2. The new key is sent to only those users in $GlobalSecret(k)$.
3. A recipient only accepts a key k if it is in the *GlobalSecret* set.
4. Each key k is received together with the set of users in $GlobalSecret(k)$. The new keys and the set of users are added to the *LocalSecret* set of the recipient.

Theorem 1 *With the above security assumptions, if Alice starts in a secure state then she remains in a secure state after receiving a new key.*

The proof of this theorem is immediate. Key generation and distribution does not affect Alice's security criterion since it involves a new key which cannot be in the domain of her local secrets. When she receives a new key she adds it to her *LocalSecret* set and maps it to the set of all users which it is mapped to in *GlobalSecret*. Thus the security criterion is automatically maintained.

In the next subsection, the problem is addressed of how Alice can be confident that the received key is in the global set.

2.2 Abstract Protocol Messages

The cryptographic keys known to system users result in a set of secure channels between pairs of users [1]. These channels provide confidentiality of data (the sender knows who will receive the message) or authentication of data (the recipient knows who has sent the message) or both. A useful notation for use of these channels is as follows.

$$A \xrightarrow{c} B : M$$

This means that A sends the message M on a confidentiality channel to B. Note that this is an action of the process representing A and implies no action for the process of B which may or may not receive such a message.

$$B \xleftarrow{a} A : M$$

Similarly this means that B receives a message M on an authentication channel from A. This is an action involving only the process of B. However, B may correctly deduce that at some time in the past M was formed by someone in possession of the key of A. This notation is now used to represent the abstract processes involved in the protocols.

In order to convey a symmetric key from one user to another (key exchange) a confidentiality and an authentication channel must both exist [1]. Such channels may need to be set up as part of the protocol. Cases where these do not initially exist will be discussed below. The format of messages used to convey keys can be derived by examining the required properties. An authorised principal S will pass a key k to a recipient A only over a confidentiality channel. This is necessary for security assumption 2.

$$S \xrightarrow{c} A : k$$

In the model we regard the generation and sending of the key k to the set $\mathcal{R}$ of all recipients to be a single state change and this is recorded as a corresponding update to the *GlobalSecret* set.

$$GlobalSecret' = GlobalSecret \cup \bigcup_{U \in \mathcal{R}} \{(k, U)\}$$

An authorised principal will only accept a key k from the server if it is received in an authenticated message. In order for the recipient to be confident that the key is indeed in the *GlobalSecret* set (security assumption 3) she must have some way of knowing that it is newly generated. For this purpose it must include a liveness indicator, or *nonce*, N which ensures that the message (and hence the key) is a new one. The key must be accompanied by a set $\mathcal{R}$ of the names of all users who are associated with k in *GlobalSecret*, in other words, the names of all users who have been sent k.

$$A \xleftarrow{a} S : k, \mathcal{R}, N$$

In the model this means that the recipient Alice must update her *LocalSecret* by adding all the pairs (k, U) for all users U in $\mathcal{R}$ (security assumption 4).

$$LocalSecret'(Alice) = LocalSecret(Alice) \cup \bigcup_{U \in \mathcal{R}} \{(k, U)\}$$

If these procedures for the sending and receiving of messages are followed by all authorised users then the security criterion is maintained. This is simply a re-interpretation of the theorem from the last subsection. Note that it is not necessary to consider what attacks may be mounted against the system. What we know is that if the authorised principals act correctly and the security channels provide the stated properties, then the security criterion will be maintained regardless of the actions of an attacker.

2.3 Concrete Protocol Messages

In a concrete protocol the conditions for both the sender and the recipient must be satisfied together. This can be done by sending messages using channels which provide both confidentiality and authentication at the same time. This is the way that protocols have usually been designed in the past. However there is no need for this restriction as long as the key is always sent along a confidentiality channel. The reason that this can be useful is that authenticated messages are very often not authenticated as plaintext, but after processing by a oneway hash function [19]. Such a oneway function may be regarded as providing a confidentiality channel to the empty set of users - there is no key to recover the message. However the oneway function is sufficient to provide specific authentication channels.

In order to describe concrete protocols some notation to record cryptographic transformations is required. This is achieved through three different kinds of brackets to distinguish between transformations which provide confidentiality, those which provide authentication, and those which provide both.

- $[\![M]\!]_k$ **confidentiality**: the message M cannot be recovered from the transformed data without knowledge of k.
  Typical transformations providing such a property are the Data Encryption Standard (DES) in one of its usual modes for data encryption [18], encryption using the public key of an asymmetric cryptosystem [17], or the unconditionally secure one time pad [20].

- $[M]_k$ **authentication**: the transformed data may not be formed from the message M without knowledge of k.
  Typical transformations providing this property are a Message Authentication Code (MAC) such as may be formed using the Data Encryption Standard (DES) [18], encryption using the private key of an asymmetric cryptosystem [17], or an unconditionally secure authentication code [21].
- $\{M\}_k$ **confidentiality and authentication**: both the above properties hold.
  Transformations providing both properties will typically have distinct elements dedicated to each property. A typical examples would be concatenation of a digest of the message using a oneway hash function [19] prior to DES encryption. Another example is a digital signature scheme followed by encryption with the public key of the recipient, but here the key k must be identified with two separate keys and the transformation may usually be considered as two distinct operations.

Using this notation the two general formats for key exchange messages are as follows.

1. $$\{k, \mathcal{R}, N\}_{k_S}$$
   where k_S is a key associated by the recipient with an authorised user and is suitable for use to provide both confidentiality to the recipient and authentication from the sender. As above, $\mathcal{R}$ is a set of recipient names and N is a nonce. This is a typical key exchange message in existing protocols. Unfortunately the requirement of both authentication and confidentiality properties is often not made explicit and this can lead to attacks [12].
2. $$[\![k]\!]_{k_A}, [h(k, \mathcal{R}, N)]_{k_S}$$
   where k_A is a key associated by the sender with an authorised user and is suitable to provide confidentiality and k_S is a key associated by the recipient with an authorised user and is suitable for use to provide authentication. This is not a typical format for key exchange messages in existing protocols. Nevertheless it has significant potential advantages over the first format. Firstly, it allows flexibility in choice of cryptographic algorithms; for example the unconditionally secure one time pad may be efficiently used for confidentiality. Secondly, it will often be more efficient than the first format, by limiting cryptographic transformations only to those fields where they are required. In some situations this latter advantage will be accompanied by a very small field size for the secure messages.

2.4 Nonces and Channel Inversion

So far it has been assumed that both confidentiality and authentication channels exist from the sender S to the recipient of the key A. Although a common situation this need not always be the case initially and it may be necessary

to invert existing secure channels. This can be done efficiently as part of the protocol. In order for key exchange to be possible the initial state must include either $S \xrightarrow{c} A$ or $S \xleftarrow{a} A$ and also either $A \xleftarrow{a} S$ or $A \xrightarrow{c} S$ [1].

Suppose, for example, that $A \xrightarrow{c} S$ exists initially but not the desired $A \xleftarrow{a} S$. Then the new channel $A \xleftarrow{a} S$ may be formed by sending a key using $A \xrightarrow{c} S$.

$$A \xrightarrow{c} S : k_A$$

As long as k_A is a key suitable for use for authentication then it may be used by S to authenticate a new key passed back to A. There is no reason why this mechanism should not be used with a subset of those users involved in the protocol. Similarly the channel $A \xrightarrow{c} S$ may be converted to $A \xleftarrow{a} S$ by passing a public key for confidentiality over the channel $A \xleftarrow{a} S$. An example of this is to send a Diffie-Hellman [5] key exchange message, just in the one direction, which establishes a confidentiality channel in the other direction.

It has also been assumed up to now that the sender of the key is able to include a nonce in the message to show that the key is new. Nonces may be classified into the three types: timestamps, counters and random challenges. The first two may be assumed to be known to the sender without any information from the recipient. The third requires a previous protocol message which is the challenge sent from key recipient to key sender. All types are equally applicable to the model. Selection and use of appropriate nonces is complex in practice [8] and is outside the scope of this paper.

3 Design of Protocols

In this section the design rules developed in the previous section will be brought together to design a number of concrete protocols.

3.1 User to User Protocols

The basic protocol using the first format is a single message as follows, specified in the traditional manner indicating only a successful message exchange.

1. $A \rightarrow B : \{k, A, B, N\}_{k_{AB}}$

Here k_{AB} is a key already shared by A and B which is suitable to provide both confidentiality and authentication. N is a nonce which is initially predictable by A, so is a timestamp or a sequence number. A implicitly sends the key to itself and hence updates its own *LocalSecret* when it sends such a message. The recipient updates its *LocalSecret* when it receives a message of the correct format, otherwise rejects it. The variation of this protocol using a random challenge as a nonce is only trivially different.

1. $B \rightarrow A : N$

2. $A \rightarrow B : \{k, A, B, N\}_{k_{AB}}$

The second type of message has its basic form as follows.

1. $A \rightarrow B : [\![k]\!]_{k_B}, [h(k, A, B, N)]_{k_A}$

Here k_B (which may be public or shared) and k_A (which may be private or shared) are keys already known by A with k_B suitable to provide confidentiality to B and k_A suitable to provide authentication from A to B. Again it is assumed that the nonce N is predictable to B and a trivial variation allows B to issue a random challenge.

1. $B \rightarrow A : N$
2. $A \rightarrow B : [\![k]\!]_{k_B}, [h(k, A, B, N)]_{k_A}$

Further variations on these two types of protocols cover the cases where the initial channels are not $A \xrightarrow{c} B$ and $B \xleftarrow{a} A$ and so require channel inversion. For example the following protocol is suitable when k_{BA} is a key initially known by B (perhaps a public key) and suitable for providing confidentiality to A.

1. $B \rightarrow A : [\![k_A]\!]_{k_{BA}}$
2. $A \rightarrow B : [\![k]\!]_{k_B}, [h(k, A, B, N)]_{k_A}$

3.2 Protocols using a Trusted Party

These protocols make use of a trusted server S which sends a key to principals A and B for use with each other. The basic protocols of the first type is as follows.

1. $S \rightarrow A : \{k, A, B, N\}_{k_{SA}}$
2. $S \rightarrow B : \{k, A, B, N\}_{k_{SB}}$

Here k_{SB} and k_{SB} are keys already shared by S and A, and by S and B respectively, which are suitable to provide both confidentiality and authentication. The basic protocol of the second type is this.

1. $S \rightarrow A : [\![k]\!]_{k_A}, [h(k, A, B, N)]_{k_{SA}}$
2. $S \rightarrow B : [\![k]\!]_{k_B}, [h(k, A, B, N)]_{k_{SB}}$

Here k_A, k_B (which may be public or shared) and k_{SA}, k_{SB} (which may be private or shared) are keys already known by S with k_A and k_B suitable to provide confidentiality to A and B respectively, and k_{SA} and k_{SB} suitable to provide authentication from S to A and B respectively.

A practical implementation will require further messages to enable principals to place messages received in context. In addition, of course, specific algorithms will need to be chosen which are believed to be either confidentiality or authentication functions or both. A particularly simple way to authenticate a hashed value is to include a secret key in the elements to be hashed.

$$[h(X)]_k = h(k, X)$$

This relies on the assumption that it is not possible to form the value $h(k, X)$ from X without knowledge of k. While this is a reasonable looking assumption, it does not follow from the usual definition of a one-way hash function and so must be explicitly stated. Using such a function an implementation might be as follows.

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : A, N_a, B, N_b$
3. $S \rightarrow B : [\![k]\!]_{k_{AS}}, h(k_{SA}, k, A, B, N_a), [\![k]\!]_{k_{BS}}, h(k_{SB}, k, A, B, N_b)$
4. $B \rightarrow A : [\![k]\!]_{k_{BS}}, h(k_{SB}, k, A, B, N_b)$

Here k_{SA} and k_{SB} are keys shared by S with A and B respectively. It is perfectly acceptable to have $k_{SA} = k_{AS}$ but it may be preferred to let k_{AS} be different, for example to be used from a one-time pad to give unconditional secrecy to the keys. This is done with maximum efficiency by applying only to the key bits.

The above protocol specification is finally in a form suitable for comparison with usual key exchange protocols. A similar implementational version can be given to all the above protocols by a choice of suitable algorithms, hints for principals, and physical paths for the messages.

3.3 A Conference Key Protocol

Since it was at no time assumed that the set of recipients $\mathcal{R}$ in the abstract protocols only had two users, the above protocols generalise naturally to allow conference key distribution. These protocols again make use of a trusted server S which sends a key to principals in the set $\mathcal{U} = \{U_1, U_2, \ldots, U_N\}$ for use with each other. The basic protocol of the first type consists of the following message sent to each U_i.

1. $S \rightarrow U_i : \{k, \mathcal{U}, N\}_{k_{SU_i}}$

As usual the k_{SU_i} are keys already shared by S and U_i suitable to provide both confidentiality and authentication. The basic protocol of the second type is the following message sent to each U_i.

1. $S \rightarrow U_i : [\![k]\!]_{k_{U_i}}, [h(k, \mathcal{U}, N)]_{k_{S_i}}$

As usual k_{U_i} and k_{S_i} are keys already known by S (which may be public, private or shared) with k_{U_i} suitable to provide confidentiality to U_i and k_{S_i} suitable to provide authentication from S to U_i.

4 Discussion

The technique of this paper appears to be the first attempt to enable design of key exchange protocols which allows flexible choice of cryptographic algorithms

relying only on their fundamental properties. An advantage of the current technique is that algorithms may be chosen appropriate to the implementation. If desired these can be unconditionally secure, or they may be a standard mode of operation for a block cipher.

Although the motivating idea has been to produce protocols which are known to be secure in terms of the defined model, it is interesting to make a comparison with existing protocols. Protocols of the first type, with combined authentication and confidentiality algorithms, are similar to a number already existing in the literature. Those using a trusted server and two other principals can be compared with those of Otway and Rees [16] and Needham and Schroeder [14]. The new protocol is similar to these but is more efficient both in terms of lengths of messages and use of encryption as well as checking required by the server. Conference key generalisations do not appear to have been discussed in the literature.

Protocols of the second type, with separate confidentiality and authentication algorithms, are by contrast not at all common in the literature. Perhaps the closest are some variations of the Diffie-Hellman key exchange protocol [6] which are again specific to a particular algorithm. These protocols have the potential to be particularly efficient in terms of lengths of messages and use of encryption and are also particularly flexible for selection of algorithms.

Despite its generality in a number of directions, the technique has a number of limitations at present and there is considerable potential for extensions. In particular:

- it has been a clear assumption that all legitimate principals act correctly. However, it seems clear that the local security of one user is not affected by the actions of any other users which are not trusted in the distribution process. It should be possible to model the effect of untrustworthy behaviour of one principal on other principals.
- more complex protocols involving distributed servers to allow authorisation [9] and delegation of rights will require a considerable further development.

References

1. C.A.Boyd, *Security Architectures using Formal Methods*, IEEE Journal on Selected Areas on Communications, June 1993, pp.694-701.
2. C.A.Boyd and W.Mao, *On a Limitation of BAN logic*, Proceedings of Eurocrypt 93, Springer-Verlag 1993.
3. M.Burrows, M.Abadi, and R.Needham, *A Logic of Authentication*, ACM Transactions on Computer Systems, Vol 8,1, February 1990, pp 18-36.
4. D.W.Davies and W.L.Price, *Security for Computer Networks*, John Wiley and Sons, 1989.
5. W.Diffie and M.E.Hellman, *New Directions in Cryptography*, IEEE Transaction on Information Theory, IT-22, pp.644-654, 1976.
6. W.Diffie, P.C.VanOorschot and M.Wiener, *Authentication and Authenticated Key Exchanges*, Designs, Codes and Cryptography, 2, pp.107-125 (1992).

7. L.Gong, R.Needham & R.Yahalom, *Reasoning about Belief in Cryptographic Protocols* Proceedings of the 1990 IEEE Computer Society Symposium on Security and Privacy, pp. 234-248, IEEE Computer Society Press, 1990.
8. L.Gong, *Variations on the Themes of Message Freshness and Replay*, IEEE Security Foundations Workshop, pp.131-136, 1993.
9. L.Gong, *Increasing Availability and Security of an Authentication Service*, IEEE Journal on Selected Areas on Communications, June 1993, pp.657-662.
10. C.B.Jones, *Systematic Software Development Using VDM*, Prentice-Hall, 1986.
11. W.Mao and C.A.Boyd *Towards Formal Analysis of Security Protocols*, IEEE Security Foundations Workshop, pp.147-158, IEEE Press, 1993.
12. W.Mao and C.A.Boyd, *On the use of Encryption in Cryptographic Protocols*, Proceedings of 4th IMA Conference on Coding and Cryptography, *to appear.*
13. C.Meadows, *A System for the Specification and Analysis of Key Management Protocols*, Proceedings of the 1991 IEEE Computer Society Symposium on Security and Privacy, pp. 182-195, IEEE Computer Society Press, 1991.
14. R.M.Needham & M.D.Schroeder, *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, 21,12, December 1978, 993-999.
15. D.M.Nessett, *A Critique of the Burrows, Abadi and Needham Logic*, ACM Operating Systems Review, 24,2,pp.35-38,1990.
16. Dave Otway & Owen Rees, *Efficient and Timely Mutual Authentication* ACM Operating Systems Review, 21,1,pp.8-10, 1987.
17. R.Rivest, A.Shamir & L.Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, 21,pp.120-126,1978.
18. M.E.Smid and D.K.Branstad, *The Data Encryption Standard: Past and Future*, Proceedings of the IEEE, 76,5,pp.550-559, 1988.
19. R.L.Rivest, *The MD4 Message Digest Algorithm*, Advances in Cryptology - CRYPTO '90, Springer-Verlag, 1991.
20. C.E.Shannon, *Communication Theory of Secrecy Systems*, Bell Systems Technical Journal, pp.656-715, 1949.
21. G.J.Simmons, *A Survey of Information Authentication*, in Contemporary Cryptology, G.J.Simmons *Ed.*, pp.379-419, IEEE Press, 1992.
22. E.Snekkenes, *Exploring the BAN Approach to Protocol Analysis*, Computer Security Foundations Workshop, pp.171-181, IEEE Press, 1991.
23. P.Syverson, *The Use of Logic in the Analysis of Cryptographic Protocols*, IEEE Symposium on Security and Privacy, pp.156-170, 1991.