User Functions for the Generation and Distribution of Encipherment Keys

R.W. Jones
International Computers Ltd.
Lovelace Road,
Bracknell, Berks,
U.K.

Abstract

It is generally accepted that data encipherment is needed for secure distributed data processing systems. It is accepted, moreover, that the enciphering algorithms are either published or must be assumed to be known to those who wish to break the security. Security then lies in the safe keeping of the encipherment keys, which must be generated and stored securely and distributed securely to the intending users.

At an intermediate level of detail of a system it may be useful to have functions which manipulate keys explicitly but which hide some of the details of key generation and distribution, both for convenience of use and so that new underlying techniques can be developed. This paper offers a contribution to the discussion. It proposes key manipulation functions which are simple from the user's point of view. It seeks to justify them in terms of the final secure applications and discusses how they may be implemented by lower level techniques described elsewhere. The relationship of the functions to telecommunication standards is discussed and a standard form is proposed for encipherment key information.

T. Beth, N. Cot, and I. Ingemarsson (Eds.): Advances in Cryptology - EUROCRYPT '84, LNCS 209, pp. 317-334, 1985.

1. Introduction

It is generally accepted that data encipherment is needed for secure distributed data processing systems. It is accepted, moreover, that the enciphering algorithms are either published or must be assumed to be known to those who wish to break the security. Security then lies in the safe keeping of the encipherment keys, which must be generated and stored securely and distributed securely to the intending users. A number of schemes have been proposed, and in some cases implemented, to manipulate keys securely. For example refs. 1, 2 and 3 describe different methods and offer different but overlapping sets of facilities to the user. It is likely that new methods will be developed and that some part of these methods should be hidden from the user. Since the subject has clearly not reached a stable point it is very likely that any attempt at present to establish a standard user interface will soon need revision. Nevertheless, this paper is written on the assumption that a discussion of such an interface is useful, since it helps to identify the common features of different schemes and to gain some idea of which features will become generic and which become part of the underlying mechanisms.

At some level the user does not concern himself with the manipulation of keys or with explicit commands to encipher and decipher data. He asks for a secure connection to another user or for a securely stored file and can assume that such details are thereby taken care of. At a lower level software and hardware logic exists which deals with things such as how keys are generated, how data encipherment keys and key encipherment keys are kept distinct and the manner of transporting a data encipherment key to a remote user.

At an intermediate level of detail it may be useful to have functions which manipulate keys explicitly but which hide some of the details, both for convenience of use and so that new underlying techniques can be developed. This paper discusses this

intermediate level. In doing so it must make assumptions about which functions are primitive at this level. For example, since a digital signature may be achieved by enciphering a message digest, using the secret member of a public key pair, one might decide that it is an application to be programmed in terms of encipherment primitives and does not give rise to specific primitive operations. This view is invalidated by signature techniques which do not depend upon encipherment. Similarly there is implicit in such an interface a judgement of which of the details which should be hidden. Ref. 4 describes a key distribution centre. In an appropriate context software at some level submits a request to a key distribution centre (KDC) for a key which can be used to communicate securely with an intended correspondent. We may wish to produce software which needs no modification when moved from such an environment to one where the system supporting the application user keeps records to enable it to issue keys securely to all members of the community. If this is so we should hide the use or non use of the KDC, but we judge in doing so that the user at that level has not lost needed flexibility. Such judgements as these are made in what follows and the reasons for them are discussed.

2. The Functions

This section describes a set of functions to generate and manipulate keys. The intention is that they appear simple to the user. The user is somewhat ill defined, but well enough, it is hoped, for the benefit of the discussion. One candidate is certainly an application process which makes use of an application service as defined in the Open Systems Interconnection model (see ref. 8) and which wishes to perform explicit data encipherment. Another candidate is the logic of a transport layer entity in the Open Systems Interconnection model which offers a secure service to users of the transport service and which, therefore, sends a data enciphering key to a remote transport entity. The functions are as follows.

i) Generate key(t,s) meaning generate for me a key or a pair of keys of type t and return to me, as the result of this function, the local name of the item containing the key or keys. The type shows, among other things, whether a symmetric or asymmetric algorithm is involved. In the former case a single key is generated and returned as the result of the function. In the latter case the enciphering and deciphering pair is generated and returned. The local name is subsequently used subscripted by 1 or 2 to indicate an individual member of a key pair thus generated or unsubscripted to mean the single key generated or the complete item containing the key pair. s is a 64 bit string, supplied by the caller, which is to be used by the key generation function. The caller does not know the cleartext value of the key generated but is assured that the same t and s values in a subsequent call generate the same key or keys. s may be omitted, in which case the values generated, as far as the caller is concerned, are random. His chance of generating them again is random. The type t is an integer. Possible meanings assigned to its values are:
a key enciphering key (KEK) for DEA1,
a data enciphering key (DEK) for DEA1.
an RSA key pair to be used for enciphering keys.
Other meanings, to which values might be assigned, are discussed in section 3.

N.B. this function and the next two have a result. The assumption is that the user has a notation which enables him to write something like

x := generate key (y, z).

The variable which is to hold the result could be written as another parameter. This is a matter of taste.

ii) Give key(k,q) meaning send my key whose local name is k securely to the user known to me as q. Assign to the key a common reference number which we may use in messages to each other and in communicating with our local encipherment services (of which this function forms a part). Make the reference number available to q

and return it to me as the result of this function. N.B. the exact manner of making it known to q that the key is available for him is not considered here. In an implementation it would not be a trivial issue. Similarly although we may assume that the services at the users' locations acknowledge receipt to each other there is need to consider whether the end user should do so as well. The assumption here is that if this is done it is separate from the basic functions needed for key distribution.

iii) Mutual key(t,q,s) meaning generate a mutual key for me and user q. Use seed s and give the key type t. t and s are as in "generate key". s may be omitted to obtain a random key. Assign to the key a common reference number and make it available to q and return it to me as the result of this function.

iv) Take key(r,q) meaning make the key whose reference number is r unavailable to user q.

v) Destroy key(K) meaning destroy the key identified by K. K may be a local name of a key, created by "generate key" or a reference number created by "give key" or "mutual key"

3. Use of the Functions

This section considers the functions of section 2 in the light of applications of encryption and related techniques.

3.1 Connection Establishment and User Authentication

When establishing a connection between two users so that they may exchange messages protected by encryption (for example if they use an insecure telecommunication link) both users (or their local services) must be provided with a key and the users must be authenticated to each other's satisfaction. "Give key" and "mutual key" may both be used to send a key to a remote user (the

reason why both exist is discussed in section 4). A reasonable requirement of either of these functions is that it delivers the key, guarantees to the initiator that the recipient is the user requested, tells the recipient from whom the key came and guarantees that he, in his turn, is who he claims to be, i.e. not just a legitimate user of the service. This is illustrated in figure 1., where A is one of a number of users of the A service and B is one of a number of users of the B service. The A service is used by A in a controlled environment in which the identity of A is assured (for example the process which represents him has been initiated after the submission of a password to a control program which controls access to resources, one of which is the A service). B has the same relationship to the B service. The route between the A service and the B service is assumed to be insecure in the absence of encipherment.

Fig. 1

After receiving a request from A to deliver a key to B the A service, having discovered the route, sends it to the B service, suitably enciphered by a KEK. The A service and the B service must authenticate each other. Their manner of doing this depends upon a number of factors,including whether a KDC is involved and whether the KEK is a public or secret key. Methods are discussed, for example, in refs.4 and 7. For example, ref. 4 describes protocols for sending a DEA1 key, first when it is protected by DEA1 encryption and secondly when it is protected by public key encryption. In both cases the protocol is described in terms of a user A who wishes to send a key to another user B, with the aid of a KDC (see fig. 2).

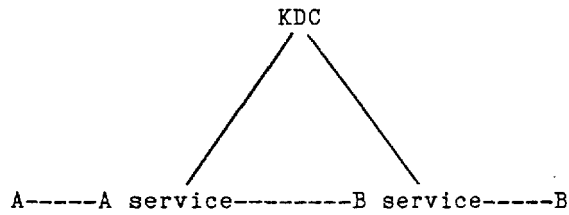

Fig.2

In the first case the protocol has three logical parts viz:

i) A obtains securely from KDC two copies of the key, one enciphered by A's KEK and the other enciphered by B's KEK.

ii) A sends to B the copy enciphered by B's KEK.

iii) A and B use the key to exchange authentication protocol.

In the second case the protocol has four logical parts viz:

i) A obtains securely from KDC B's public key and the key to be used.

ii) A sends to B the key enciphered by B's public key.

iii) B obtains securely from KDC A's public key.

iv) A and B exchange authentication protocol.

(For details of the values exchanged to cope with particular security problems see ref. 4.)

Either of these methods may be hidden from the users at the level proposed for them here. The appropriate interchanges are initiated by the function 'mutual key'. A possible improvement in underlying protocols to remove as yet unknown security flaws is also hidden from them.

Once the two services have authenticated each other they may

trust each other to have authenticated the users they serve and therefore to give A and B a service which authenticates the remote user.

Having obtained a mutual key, the two users, if they are particularly suspicious, may wish to exchange further messages to convince themselves of each other's genuineness. This must depend upon further secret information, which becomes vulnerable if it is sent to the other,as yet untrusted, party, using the newly established connection. They may, for example, exchange passwords using the protection of the connection they do not quite trust. If a correct reply password is not received within the permitted number of attempts the first one is compromised and there is a suspicion that the key distribution service is in error. The users may, on the other hand, have private encipherment keys, previously delivered, which they use only to protect their private authentication protocol. If the protocol reveals a doubt of correct identity no secret user information is compromised but, as before, the trustworthiness of the key distribution service is in doubt. This kind of consideration is inevitable if there is a standard service which distributes keys and attempts to guarantee that the sender and recipient are genuine. An alternative is that the service does not use encipherment to authenticate the users, but leaves it to them. Another is that the identity of the recipient is guaranteed but that he is only sure that the originator is an authorised user of the key distribution service. Neither of these possibilities seems as useful since one or both users must either risk compromising secret information or must hold a key personally. They may well do so but they should not be forced to.

Another point to consider is that a user who wishes to connect to a remote resource may not be directly identifiable by that resource. For example, a database interrogation service may contain no check of its user's authority, assuming that his identity was established as part of the identification procedure when he logged in and that the resources at his disposal, including the interrogation service, were thereby decided. There will then be an entity, at the same location as the user who wishes to connect, which is concerned with resource allocation, which knows which users are allowed to use which resources and which checks permission before allowing the user's connection to

be made. This entity has a privileged position in remote user authentication in that it is trusted by remote parts of the service (entities of the same kind as itself) to guarantee that the users it serves are only given authorised connections. It is useful to build into the service some mechanism to guarantee to such privileged entities that they are communicating with their own kind. The simplest way of doing this is to design the control software so that all connections to remote processes are handled by such entities and that they check access permission at one or both of the sites involved. If we assume that this is not the case and that there is a need to make connections between processes which will do their own checking of authorisation then a possible way of identifying the entities which are to be given more trust is to allocate exclusively for their use a special type of key. The encipherment service guarantees to the remote encipherment service that such a key may only be used successfully by such an entity. Ref. 2 introduces the idea of type values which it is useful to bind securely to keys (e.g. DEK or KEK). A useful type value which is not mentioned there is one which guarantees that the key may be used only by an entity authorised to check access rights.

There are applications where it is useful to be able to generate the same key at two remote sites rather than sending the key from one to the other and without sending values used to generate it via the telecommunication link. For example, a customer is supplied with a plastic card which is used to help identify him. The card contains a value which is to help generate the key to be used in sending information to a central installation. In addition he is required to type in a PIN value which also contributes. Another contributory value comes from the terminal into which he inserts his card (the terminal value may be changed periodically for greater security). The central installation holds these values. When it is told in clear who the customer and the terminal claim to be it generates a key using the stored values, knowing that the genuine terminal can generate the same on behalf of the genuine user. For this and similar cases the key generation functions in section 2 contain a seed value, with the assurance that the same seed will generate the same key. When an unrepeatable key is wanted the seed is omitted. There is, of course, a danger in this facility and it may well be that it should be denied to some users.

In making a request for a transport connection, as described in the Open Systems Interconnection model, it is envisaged that a user may ask that it be secure. The details of what this means are not yet spelled out but it certainly implies encipherment. A connection request message may contain 'security parameters' (see ref. 9) and we may suppose that they will indicate the key to be used, either as the actual key (suitably enciphered) or as a reference to a key already known to both parties. We may then consider the applicability of the functions described here. First if the two parties have an established mutual KEK used to encipher keys they wish to send each other the functions are not applicable. The key to be used for the connection is enciphered by a call on the sender's encipherment service. It may then either be placed in the connection request message or it may be sent beforehand (for example as one of a batch of keys to use that day) and a reference to it may be placed in the connection request message. If the two parties do not have such a mutual KEK and do not have a supply of session keys to choose from then the function 'mutual key' applies. However, it cannot be used to encipher the key which is then placed in the connection request because that is not its function. Its function is to deliver the key. Neither is it reasonable to suppose that a key should be extracted from the connection request as it passes from one KEK domain to another (and there may be such separate domains for security purposes). The use of 'mutual key' in this case is to establish a mutual key for the two end users so that they may use it to encipher the keys to be used subsequently for transport connection protection. It must be done as a separate previous operation and, at least the first time, must be sent over an 'insecure' transport connection. This does not matter as the function handles its own security.

3.2 **Data Privacy & Data Authentication**

Once keys have been successfully exchanged by the two end users of a telecommunication link or by their local services on their behalf data privacy may be achieved by data encipherment and decipherment. Each local service must therefore provide enciphering and deciphering functions. The user may also wish to encipher and decipher keys using key enciphering keys to produce and make use of key hierarchies. These topics are dealt with for

example in refs.1 and 2, which describe means of protecting keys such that they never appear in clear outside a trusted encipherment environment. They are relevant to this paper in that the user of the key manipulation facility needs the ability to operate explicitly upon keys of a chosen type, but should not need to know how the types are indicated or need to be wary of operations upon keys of a particular type which might prejudice security. Data authentication and greater assurance of privacy are obtained by using particular modes of operation of encipherment (for example cipher block chaining or cipher feed back when using block ciphers) and by the addition of checking information (for example enciphered sum checks to reveal illicit modification and various identifying values to reveal illicit insertions and replays). These functions are not directly concerned with key generation and distribution and are not dealt with in this paper.

3.3 Digital Signatures

A digital signature depends upon a sender using a key that no one else has and the receiver being able to demonstrate that the key has been used. To do this the sender may use the secret key of a public key cipher, such as RSA, and make the public key available to the receiver (ref.5). Using the functions described here a type value would be assigned to mean a public key pair. The effect of a public key cipher may be achieved by adding type information, meaning "encipher only" or "decipher only" to a symmetric cipher key in a trusted environment, with the knowledge that it can only be removed and acted upon in a trusted environment (ref.2). Another possibility is to use an algorithm which has an associated public and private key but which transforms the text to be signed by some means other than encipherment. Such keys can also be indicated by type information in the functions described in section 2.

3.4 Stored Secure Files

The key generation function may be used to generate a key which enciphers a file stored locally or whose medium is to be physically removed from the computer environment. If a file is stored for a long time or is transferred to a separate site it will be necessary to re-encipher. Ref.1 points out that a

hierarchy of keys is needed in such a case. Refs. 1 and 2 discuss how this may be achieved securely. The exact method is hidden at a lower level and visible in the functions described here only in the fact that keys are generated with an explicit type which indicates Key Enciphering Key or Data Enciphering Key.

3.5 Protection of Software Copyright

Ref.2 points out that type information securely attached to a key may be used, given a secure execution environment, to safeguard copyright. Software to be protected would be enciphered by the key and the key would be supplied to the user enciphered by a KEK which was available only inside the secure execution environment. When the software was used it would be deciphered as an implicit part of the loading operation. This idea anticipates the commercial availability of such an execution environment. However, when appropriate, a type value could be assigned in the functions of section 2.

4. Relationship to Detailed Key Manipulation Schemes

This section discusses how the functions described in section 2 can be implemented using a number of techniques described elsewhere. The functions are dealt with in turn.

4.1 Generate key

Let us assume we are using one of the key management schemes described in refs.1, 2, and 3. Each of them, when it generates a key and makes it available outside the trusted encipherment facility protects it by enciphering it. The schemes differ in how they do this and in how they ensure that the keys may not be misused (for example that a DEK may not be deciphered and made available outside the encipherment facility in clear form). They differ in the amount of protection they give the keys. The Key Notarization Scheme guarantees that a key can only be used successfully by the intended users by making the encipherment and decipherment of the key a function of the identities of the users

for whom the key is intended. Since a user must establish his identity in a way which satisfies security criteria (for example by supplying a password) he cannot successfully use someone else's key. The IBM scheme protects the key from exposure and ensures that some different types of key cannot be confused. To do this different master keys at an instalation are used to encipher KEKs, session keys and keys used to encipher files. The operating system is relied upon to ensure that the keys are used by the intended users. The ICL scheme enciphers a key, together with type information indicating how it may be used, by a KEK (in some cases by an instalation master key). It can, therefore, potentially restrict keys in ways which may be defined and could include the equivalent of the Key Notarization scheme. The functions supplied in terms of key type therefore overlap and where they coincide they are not implemented in the same way. The functions described in section 2 may be mapped on to any of the three, with the proviso that some of the key types envisaged are not present in some cases.

The local name produced by "generate key" is then in the context of ref.1 the form enciphered by KM0, KM1 or KM2 according to its type. In the context of ref.2 it is the key and concatenated type enciphered by the master key. In the context of ref.3 it is the form supplied by the Key Notarisation Facility.

If a key is to be associated securely with its users as in ref.3 then extra associated software is needed if the basic encipherment facility does not provide it. Whether it is always desirable to tie a generated key immediately to particular users is a debatable point.

4.2 **Give key**

Assume that the user to whom the key is to be given is at a site which uses a similar system in terms of refs.1, 2 and 3. If the first site has the necessary KEK it can re-encrypt the generated key and send it directly to the second site. There the service re-encrypts it for the second user if the key used to protect it in transit is not the one which protects it when it is stored there. There may, on the other hand be a series of re-encipherments en route because of the need to cross different key domains. The user of the "give key" function may remain unaware

of this.

As in ref.4., a Key Distribution Centre may be used to generate the key in a form suitable for transmission to another site. This also may be hidden from the user of the "give key" function.

If the sender and recipient are encipherment services which differ in the way they encode keys for protection (as in refs. 1, 2, and 3) more manipulation is needed to effect the transfer. There must be a transformation function, which operates in an environment as secure as the one used to encipher the data in the first place, which deciphers and re-enciphers, reformatting as necessary. This also can be hidden from the user of "give key", although a standard way of formatting keys and their associated information is clearly desirable.

4.3 **Mutual key**

In some cases this may be only a shorthand way of writing "generate key", followed by "give key". However consider the following cases.

a) When a KDC is used to generate the key it may be necessary to tell it the identity of the other partner in the connection so that it may encipher it appropriately (see, for example, ref.4).

b) The generation of the key may need the involvement of the encipherment services at both ends of the connection (for example when using the Diffie/Hellman algorithm (ref. 6).

For such reasons"mutual key" is needed as a primitive function at this level.

4.4 **Take key and Destroy key**

If the underlying implementations are those of refs. 1, 2 or 3 these functions are barely necessary. If a generated key is stored by the encipherment service and a reference to it passed back to the user then an explicit destruction of keys is needed. "Take key" may also be used to inform the service that a particular user is no longer entitled to use a key.

5. Relationship to Communication Standards

We may expect the emerging Open Systems Interconnection standards to provide secure services. For example, as already mentioned, an enhancement of the transport service is likely to provide authentication of users, data privacy and data authentication. The two entities which communicate to provide this service must establish jointly agreed keys and initialisation variables and would make use of functions such as those described in this paper. The form of the transmitted key and its accompanying information is an obvious candidate for standardisation and would avoid the need to transform the key en route, other than to change its key encryption key. In seeking a standard form we have to consider:

i) the length of the key,

ii) the permitted users (if this is to be declared explicitly),

iii) information about the type of use permitted.

The methods referred to in this paper do not all allow the same restrictions of key use to be described. Moreover, in some cases, the restriction is implied in the manner of enciphering the key (e.g. the Key Notarization scheme). A standard which explicitly stated the users could therefore be considered redundant in this case. However, if the basic key manipulation method does not involve the user's identity (as in ref.1 and in ref.2 in its simplest form) the addition gives added security.

The basic encipherment algorithm affects both the length of the key and the type information which is relevant. For example, an indication of "encipherment" or "decipherment" is irrelevant to an RSA key.

Ref. 2 has suggested that the "parity" bits in the DES key could be used to indicate typing information. This may be unacceptable as an international standard. The typing information must then be held separately from the 64 bit key variable.

Bearing these points in mind the following is a tentative suggestion for a standard form for a key and associated information. First, the clear form. It has the format:

key length, key, key type, users .

where "key length" is an integer which gives the length of the following key;

where "key" is the key as a binary string;

where "key type" is a binary string whose bits have the following significance:

1st bit	DEK or KEK,
2nd bit	enciphering key or not,
3rd bit	deciphering key or not,
4th bit	software protection key or not,
5th bit	key usable by any process or only by one authorised to check access rights,

(meanings for other bits are likely to prove useful);

and where "users" consists of either one or two alphanumeric strings which identify the permitted user or users.

If such a composite item is to be transmitted over an insecure telecommunication line it must be enciphered. The form this takes depends upon the enciphering method. Using a 64 bit block cipher, for example, one must use some method of ensuring that the separate blocks which form the item cannot be changed unnoticed. One might, for example, form an enciphered sum check of the whole item and send it with it. A method which enciphered a block as long as the composite item could dispense with this.

6. Conclusions

This paper has discussed a number of issues related to the standardisation of the interface to an enciphering service at a particular level.

Several ways of providing basic key manipulation features have been considered. It would be logically possible to evolve a standard way which made use of the best features of those considered. This would make standardisation of the form of the key and associated information easier.

An enciphering service may or may not make use of a separate Key Distribution Centre, depending on the number of communicating locations and the complexity possible in each. This design option is likely to survive. The functions suggested here deliberately hide this choice, taking the view that it is a part of the service implementation which the user should be able to ignore.

When a key is sent to a remote user it may need to be transformed because a different way of protecting it is needed. It may need to be enciphered by the remote user's location master key. During its journey it may need to be enciphered by a KEK used only for transportation. It may need to be re-enciphered by several such keys in the course of its journey. Such transformations should be hidden from the user at as low a level as possible so that logic can be written irrespective of the context created by the way the network of users is organised.

New methods of enciphering are likely to be developed. We should attempt to protect users from the need to know the underlying changes they bring. This is, of course, an aim which cannot necessarily be fulfilled. At the level chosen for the functions of this paper we reveal the essential difference between symmetric and asymmetric ciphers. New methods may bring their own characteristics which should not be hidden.

New applications of encipherment and related techniques are likely. Two mentioned here are digital signatures which do not use encipherment of a form which can be used for data privacy and a new key type dedicated to controlling resource use.

For such reasons the subject is one which will continue to develop and the points made in this paper are offered as part of the discussion needed to find functions and techniques which may develop as our knowledge of the subject grows.

References

1. Ehrsam W.F., Matyas S.M., Meyer C.D. and Tuchman W.L. : "A cryptographic key management scheme for implementing the data encryption standard." IBM Systems Journal, vol.17, no.2.

2. Jones R.W. : "Some techniques for handling encipherment keys." ICL Technical Journal, vol.3, no.2.

3. Smid M.E. : "A key notarization system for computer networks. "NBS Special Publication 500-54, US Dept. of Commerce.

4. Price W.L. & Davies D.W. : "Issues in the design of a key distribution centre." NPL Report DNACS 43/81, National Physical Laboratory, Teddington, Middlesex, UK

5. Rivest R.L., Shamir A and Addleman L. "A method of obtaining digital signatures and public key cryptosystems." Communications of the ACM, February 1978.

6. Diffie W and Hellman M.E. "New directions in Cryptography." IEEE Transactions on Information Theory, vol.IT-22, no.6.

7. Needham R.M. & Schroeder M.D. "Using encryption for authentication in large networks of computers." Communications of the ACM, December 1978.

8. International standard ISO/IS 7498. Information processing systems -Open systems interconnection - Basic reference model.

9. Draft International Standard ISO/DIS 8073. Information processing systems -Open systems interconnection - Connection oriented transport protocol specification.