Encryption and Secure Computer Networks

GERALD J. POPEK AND CHARLES S. KLINE

University of California at Los Angeles, Los Angeles, California 90024

There is increasing growth in the number of computer networks in use and in the kinds of distributed computing applications available on these networks This increase, together with concern about privacy, security, and integrity of information exchange, has created considerable interest in the use of encryption to protect information in the networks

This survey is directed at the reader who is knowledgeable about various network designs and who now wishes to consider incorporating encryption methods into these designs. It is also directed at developers of encryption algorithms who wish to understand the characteristics of such algorithms useful in network applications.

Key management, network encryption protocols, digital signatures, and the utility of conventional- or public-key encryption methods are each discussed. A case study of how encryption was integrated into an actual network, the Arpanet, illustrates many issues present in the design of a network encryption facility.

Keywords and Phrases: computer networks, computer security, encryption, public-key cryptosystems, digital signatures, network registries, encryption protocols

CR Categories. 3 9, 4 35, 4.39, 5 39, 6 29

INTRODUCTION

It has long been observed that as the cost per unit of equivalent computation in small machines became far less than in large centralized ones, and as the technology of interconnecting machines matured, computing would take on a more and more distributed appearance. This change of course is now happening. In many cases, users' data manipulation needs can be served by a separate machine dedicated to the single user, connected to a network of integrated databases. Organizational needs, such as easy incremental growth and decentralized control of computing resources and information, are also well served in this manner. Multiprogramming of general application software diminishes in importance in such an environment.

This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-C-0211.

As a result, the nature of the protection and security problem is beginning to change. Concern over the convenience and reliability of central operating system protection facilities is transferring to analogous concerns in networks. The issues of protection in computer networks differ in several fundamental ways from those of centralized operating systems. One of the most important distinctions is the fact that the underlying hardware cannot in general be assumed secure. In particular, the communication lines that comprise the network are usually not under the physical control of the network user. Hence no assumptions can be made about the safety of the data being sent over the lines. Further, in current packet-switched networks [KIMB75] the software in the switches themselves is typically quite complex and programmed in assembly language; one cannot say with certainty that messages are delivered only to the intended recipients.

CONTENTS

The only general approach to sending and storing data over media which are not safe is to use some form of encryption. Suitable encryption algorithms are therefore a prerequisite to the development of secure networks. Equally important questions concern the integration of encryption methods into the operating systems and applications software which are part of the network. We focus here on these latter issues, taking a pragmatic, engineering perspective toward the problems which must be settled in order to develop secure network functions. Cases where the safety of the entire network can be assumed are not discussed here because in these cases the problems are not special to networking.

In networks, as in operating systems, there are several major classes of protection policies that one may wish to enforce. The most straightforward policy, satisfactory for most applications, concerns data security: ensuring that no unauthorized modification or direct reference of data takes place. Highly reliable data security in networks today is feasible; suitable methods to attain this security are outlined in the later sections.

A more demanding type of policy is the enforcement of confinement in the network: preventing unauthorized communication through subtle methods, such as signaling via noticeable variations in performance [LAMP73]. One commonly mentioned (and fairly easily solved) confinement problem is traffic analysis: the ability of an observer to determine the various flow patterns of message movement. However, evidence to be presented later indicates that the conditions under which confinement in general can be provided in a network are quite limited.

In the following sections we describe problems and alternative solutions in the design of secure networks, discuss their utility with respect to data security and confinement, and present an illustrative case study. The material is intended as a practicum for those concerned with the development of secure computer networks or those who wish to understand the characteristics of encryption algorithms useful in network applications.

The Environment and Its Threats

A network may be composed of a wide variety of nodes interconnected by transmission media. Some of the nodes may be large central computers; others may be personal computers or even simple terminals. The network may contain some computers dedicated to switching message traffic from one transmission line to another, or those functions may be integrated into general-purpose machines which support user computing. One of the important functions of computer networks is to supply to users convenient private communication channels similar to those provided by common carriers. The underlying transmission me-

dia, of course, may be point to point or broadcast. Considerable software is typically present to implement the exchange of messages among nodes. The rules or *protocols* governing these message exchanges form the interface specifications between network components. These protocols can significantly affect network security concerns, as will be seen later. In any event, because of the inability to make assumptions about the communication links and switching nodes, one typically must expect malicious activity of several sorts.

1) *Tapping of Lines.* While the relevant methods are beyond the scope of this discussion, it should be recognized that it is frequently a simple matter to record the message traffic passing through a given communications line without detection by the participants in the communication [WEST70]. This problem is present whether the line is private, leased from a common carrier, or part of a broadcast satellite channel.

2) *Introduction of Spurious Messages.* It is often possible to introduce invalid messages with valid addresses into an operating network, and to do so in such a way that the injected messages pass all relevant consistency checks and are delivered as if the messages were genuine.

3) *Retransmission of Previously Transmitted Valid Messages.* Given that it is possible both to record and introduce messages into a network, it is therefore possible to retransmit a copy of a previously transmitted message.

4) *Disruption.* It is possible that delivery of selected messages may be prevented: Portions of messages may be altered, or complete blockage of communications paths may occur.

Each of the preceding threats can, in the absence of suitable safeguards, cause considerable damage to an operating network, to the extent of making it useless for communication. Tapping of lines leads to loss of privacy of the communicated information. Introduction of false messages makes reception of any message suspect. Even retransmission of an earlier message can cause considerable difficulty in some circumstances. Suppose the message is part of the sequence by which two parties communicate their identity to one another. Then it may be possible for some node to falsely identify itself in cases where the valid originator of the message was temporarily out of service.

More and more applications of computer networks are becoming sensitive to malicious actions. Increased motivation to disturb proper operation can be expected: Consider the attention that will be directed at such uses as military command and control systems (by which missile firing orders are sent), or commercial electronic funds transfer systems (with daily transactions worth hundreds of billions of U.S. dollars).

Operational Assumptions

In this paper the discussion of protection and security in computer networks is based on several underlying assumptions:

1) Malicious attacks, including tapping, artificial message injection, and disruption, are expected.
2) The insecure network provides the only available high-bandwidth transmission paths between those sites which wish to communicate in a secure manner.[1]
3) Reliable private communication is desired.
4) A large number of separately protected logical channels are needed, even though they may be multiplexed on a much smaller number of physical channels.
5) High-speed inexpensive hardware encryption units are available.

It is believed that these assumptions correctly mirror many current and future environments. In the next sections we outline properties of encryption relevant to network use. Those interested in a deeper examination should see the companion papers in this issue [LEMP79, SIMM79]. After this brief outline, the discussion of network security commences in earnest.

---

[1] It will turn out that *some* presumed secure and correct channel will be needed to get the secure data channel going, although the preexisting secure channel can be awkward to use, with high delay and low bandwidth. Distribution of the priming information via armored truck might suffice, for example.

1. ENCRYPTION ALGORITHMS AND THEIR NETWORK APPLICATIONS

1.1 Conventional Encryption

Encryption provides a method of storing data in a form which is unintelligible without the "key" used in the encryption. Basically, conventional encryption can be thought of as a mathematical function,

$$\mathbf{E} = F(\mathbf{D}, K),$$

where **D** is data to be encoded, K is a key variable, and **E** is the resulting enciphered text. For F to be a useful function, there must exist an F', the inverse of F,

$$\mathbf{D} = F'(\mathbf{E}, K)$$

which, therefore, has the property that the original data can be recovered from the encrypted data if the value of the key variable originally used is known.

The use of F and F' is valuable only if it is impractical to recover **D** from **E** without knowledge of the corresponding K. A great deal of research has been done to develop algorithms which make it virtually impossible to do so, even given the availability of powerful computer tools.

The strength of an encryption algorithm is traditionally evaluated using the following assumptions. First, the algorithm is known to all involved. Second, the analyst has available to him a significant quantity of encrypted data and corresponding cleartext (i.e., the unencrypted text, also called plaintext). He may even have been able to cause messages of his choice to be encrypted. His task is to deduce, given an additional unmatched piece of encrypted text, the corresponding cleartext. All of the matched text can be assumed to be encrypted through the use of the same key which was used to encrypt the unmatched segment. The difficulty of deducing the key is directly related to the strength of the algorithm.

F is invariably designed to mask statistical properties of the cleartext. Ideally the probability of each symbol of the encrypted character set appearing in an encoded message **E** ideally is to be equal. Further, the probability distribution of any pair (digram) of such characters is to be flat. Similarly, it is desirable that the n-gram probability distribution be as flat as possible for each n. This characteristic is desired even in the face of skewed distributions in the cleartext, for it is the statistical structure of the input language, as it "shows through" to the encrypted language, which permits cryptanalysis.

The preceding characteristics, desirable from a protection viewpoint, have other implications. In particular, if any single bit of a cleartext message is altered, then the probability of any particular bit being altered in the corresponding message is approximately $\frac{1}{2}$. Conversely, if any single bit in an encrypted message is changed, the probability is approximately $\frac{1}{2}$ that any particular bit in the resulting decrypted message has been changed [FEIS75]. This property follows because of the necessity for flat n-gram distributions. As a result, encryption algorithms are excellent error detection mechanisms, as long as the recipient has any knowledge of the original cleartext transmission.

The strength of an encryption algorithm is also related to the ratio of the length of the key to the length of the data. Perfect ciphers that completely mask statistical information require keys of lengths equal to the data they encode. Fortunately, currently available algorithms are of such high quality that this ratio can be small; as a result, a key can be often reused for subsequent messages. That is, subsequent messages essentially extend the length of the data. It is still the case that keys need to be changed periodically to prevent the ratio from becoming too small, and, thus, the statistical information available to an analyst too great. The loss of protection which would result from a compromised key is thus also limited.

1.2 Public-Key Encryption

Diffie and Hellman [DIFF76b] proposed a variation of conventional encryption methods that may, in some cases, have certain advantages over standard algorithms. In their class of algorithms there exists

$$\mathbf{E} = F(\mathbf{D}, K),$$

as before, to encode the data, and

$$\mathbf{D} = F'(\mathbf{E}, K')$$

to recover the data. The major difference is that the key K' used to decrypt the data is not equal to, and is impractical to derive from, the key K used to encode the data. Presumably there exists a pair generator which, on the basis of some input information, produces the matched keys K and K' with high strength (i.e., resistance to the derivation of K' given K, **D**, and matched $\mathbf{E} = F(\mathbf{D}, K)$).

Many public-key algorithms have the property that either F or F' can be used for encryption, and both result in strong ciphers. That is, one can encode data using F' and decode using F. The RSA algorithm is one that has this property [RIVE77a]. The property is useful in both key distribution and "digital signatures" (the electronic analogs of handwritten signatures) and will be assumed here.

The potential value of such encryption algorithms lies in some expected simplifications in initial key distribution, since K can be publicly known; hence the name public-key encryption. There are also simplifications for digital signatures. These issues are examined further in Sections 3 and 9. Rivest et al. and Merkle and Hellman have proposed actual algorithms which are believed strong, but they have not yet been extensively evaluated [RIVE77a, HELL78].

Much of the remaining material in this survey is presented in a manner independent of whether conventional- or public-key-based encryption is employed. Each case is considered separately when significant.

1.3 Error Detection and Duplicate or Missing Blocks

Given the general properties of encryption as already described, it is an easy matter to detect (but not correct) errors in encrypted messages. A small part of the message must be redundant, and the receiver must know in advance the expected redundant part of the message. In a block with k check bits, the probability of an undetected error upon receipt of the block is approximately $1/(2^k)$, for reasonably sized blocks, if the probabilistic assumption mentioned in Section 1 is valid. For example, if three 8-bit characters are employed as checks, the probability of an undetected error is less than $1/(2^{24})$ or $1/10^7$.

In the case of natural language text, no special provisions need necessarily be made, since that text already contains considerable redundancy and casual inspection permits error detection with very high probability. The check field can also be combined with information required in the block for reasons other than encryption. In fact, the packet headers in most packet-switched networks contain considerable highly formatted information, which can serve the check function. For example, duplicate transmitted blocks may occur either because of a deliberate attempt or through abnormal operation of the network switching centers. To detect the duplication, it is customary to number each block in order of transmission. If this number contains enough bits and the encryption block size matches the unit of transmission, the sequence number can serve as the check field.

Feistel et al. [FEIS75] describe a variant of this method, called block chaining, in which a small segment of the preceding encrypted block is appended to the current cleartext block before encryption and transmission. The receiver can therefore easily check that blocks have been received in proper order by making the obvious check. However, if the check fails, he cannot tell how many blocks are missing. In both of these cases, once a block is lost and not recoverable by lower level network protocols, some method for reestablishing validity is needed. One method is to obtain new matched keys. An alternative (essential for public-key systems) is to employ an authentication protocol (as described in Section 2) to choose a new valid sequence number or data value to restart block chaining.

1.4 Block Versus Stream Ciphers

Whether an encryption method is a block or stream cipher affects the strength of the algorithm and has implications for computer use. A stream cipher, in deciding how to encode the next bits of a message, can use the entire preceding portion of the message, as well as the key and the current bits. A block cipher, on the other hand, encodes each successive block of a message on the

basis of that block only and the given key. It is easier to construct strong stream ciphers than strong block ciphers. However, stream ciphers have the characteristic that an error in a given block makes subsequent blocks undecipherable. In many cases either method may be satisfactory, since lower level network protocols can handle necessary retransmission of garbled or lost blocks. Independent of whether a block or stream cipher is employed, some check data, as mentioned in Section 1.2, are still required to detect invalid blocks. In the stream cipher case, when an invalid block is discovered after decoding, the decryption process must be reset to its state preceding the invalid block.

Stream ciphers are less acceptable for computer use in general. If one wishes to be able to update portions of a long encrypted message (or file) selectively, then block ciphers permit decryption, update, and reencryption of the relevant blocks alone, while stream ciphers require reencryption of all subsequent blocks in the stream. So block ciphers are usually preferred. The Lucifer system [FEIS73] is a candidate as a reasonably strong block cipher. Whether or not the National Bureau of Standards' Data Encryption Standard (DES), with its 56-bit keys, is suitably strong is open to debate [DIFF77], but it is being accepted by many commercial users as adequate [NBS77].

1.5 Network Applications of Encryption

Four general uses of encryption having application in computer networks are briefly described in this section. Much of the remainder of this paper is devoted to detailed discussion of them.

1.5.1 *Authentication*

One of the important requirements in computer communications security is to provide a method by which participants in the communication can identify one another in a secure manner. Encryption solves this problem in several ways. First, possession of the right key is taken as prima facie evidence that the participant is able to engage in the message exchanges. The transmitter can be assured that only the holder of the key is able to send or receive transmissions in an intelligible way.

Even using secure authentication, one is still subject to the problems caused by lost messages, replayed valid messages, and the reuse of keys for multiple conversations (which exacerbates the replay problem). A general authentication protocol which can detect receipt of previously recorded messages when the keys have not been changed is presented later. The actual procedures by which keys are distributed in the general case are, of course, important, and will be discussed in subsequent sections.

1 5.2 *Private Communication*

The traditional use of encryption has been in communications where the sender and receiver do not trust the transmission medium, be it a hand-carried note or megabytes shipped over high-capacity satellite channels. This use is crucial in computer networks.

1.5.3 *Network Mail*

In the private communication function, it is generally understood that first, all parties wishing to communicate are present, and second, they are willing to tolerate some overhead in order to get the conversation established. A key distribution algorithm involving several messages and interaction with all participants would be acceptable. In the case of electronic mail, which typically involves short messages, it may be unreasonable for the actual transmission to require such significant overhead. Mail should not require that the receiver actually be present at the time the message is sent or received. Since there is no need for immediate delivery, it may be possible to get lower overhead at the cost of increased queuing delays.

1.5.4 *Digital Signatures*

The goal here is to allow the author of a digitally represented message to "sign" it in such a fashion that the "signature" has properties similar to an analog signature written in ink for the paper world. Without a suitable digital signature method, the growth of distributed systems may be seri-

ously inhibited, since many transactions, such as those involved in banking, require a legally enforceable contract.

The properties desired of a digital signature method include the following:

1) Unforgeability. Only the actual author should be able to create the signature.
2) Authenticity. There must be a straightforward way to demonstrate conclusively the validity of a signature in case of dispute, even long after authorship.
3) No repudiation. It must not be possible for the author of signed correspondence to subsequently disclaim authorship.
4) Low cost and high convenience. The simpler and lower cost the method, the more likely it will be used.

1.6 Minimum Trusted Mechanism; Minimum Central Mechanism

In all the functions presented in Section 1.5, it is desirable that there be a minimum number of trusted mechanisms involved [POPE74b]. This desire occurs because the more mechanism, the greater the opportunity for error, either by accident or by intention (perhaps by the developers or maintainers). One wishes to minimize the involvement of a central mechanism for analogous reasons. This fear of large complex and central mechanisms is well justified, given the experience of failure of large central operating systems and data management systems to provide a reasonable level of protection against penetration [POPE74a, CARL75]. Kernel-based approaches to software architectures have been developed to address this problem; they have as their goal minimization of the size and complexity of trusted central mechanisms. For more information about such designs, see MCCA79, POPE79, DOWN79.

Some people are also distrustful that a centralized governmental communication facility, or even a large common carrier, can ensure privacy and other related characteristics. These general criteria are quite important to the safety and credibility of whatever system is eventually adopted. They also constrain the set of approaches that may be employed.

1.7 Limitations of Encryption

While encryption can contribute in useful ways to the protection of information in computing systems, there are a number of practical limitations to the class of applications for which it is viable. Several of these limitations are discussed below.

1.7.1 *Processing in Cleartext*

Most of the operations that one wishes to perform on data, from simple arithmetic operations to the complex procedure of constructing indexes to databases, require that the data be supplied in cleartext. Therefore, the internal controls of the operating system, and to some extent the applications software, must preserve protection controls while the cleartext data are present. While some have proposed that it might be possible to maintain the encrypted data in main memory and have them decrypted only upon loading into CPU registers (and subsequently reencrypted before storage into memory), there are serious questions as to the feasibility of this approach [GAIN77]. The key management facility required is nontrivial, and the difficulties inherent in providing convenient controlled sharing seem forbidding. Another suggestion sometimes made is to use an encoding algorithm which is homomorphic with respect to the desired operations [RIVE78]. Then the operation could be performed on the encrypted values, and the result can be decrypted as before. Unfortunately, known encoding schemes with the necessary properties are not strong algorithms, nor is it generally believed that such methods can be constructed.

Therefore, since data must be processed in cleartext, other means are necessary to protect data from being compromised by applications software while the data are under control of the operating system, and the remarks in the previous section concerning minimization of these additional means are very important to keep in mind.

1.7 2 *Revocation*

Keys are similar to simple forms of *capabilities*, which have been proposed for operating systems [DENN66, FABR74]. They act as tickets and serve as conclusive evi-

dence that the holder may access the corresponding data. Holders may pass keys, just as capabilities may be passed. Methods for selective revocation of access are just as complex as those known for capability systems [FABR74]. The only known method is to decrypt the data and reencrypt with a different key. This action invalidates all the old keys and is obviously not very selective. Hence new keys must be redistributed to all those for whom access is still permitted.

1.7.3 *Protection Against Modification*

Encryption by itself provides no protection against inadvertent or intentional modification of the data. However, it can provide the means of detecting that modification by including as part of the encrypted data a number of check bits. When decryption is performed, if those bits do not match the expected values, then the data are known to be invalid.

Detection of modification, however, is often not enough protection. In large databases, for example, it is not uncommon for very long periods to elapse before any particular data item is referenced. It is only at this point that a modification would be detected. Error correcting codes could be applied to the data after encryption in order to provide redundancy. However, these will not be helpful if a malicious user has succeeded in modifying stored data and has destroyed the adjacent data containing the redundancy. Therefore, very high quality recovery software would be necessary to restore the data from (possibly very old) archival records.

1 7 4 *Key Storage and Management*

Every data item that is to be protected independently of other data items requires encryption by its own key. This key must be stored as long as it is desired to be able to access the data. Thus, to be able to protect a large number of long-lived data items separately, the key storage and management problem becomes formidable. The collection of keys immediately becomes so large that safe system storage is essential. After all, it is not practical to require a user to supply the key when needed, and it is not even practical to embed the keys in applications software, since that would mean the applications software would require very high quality protection.

The problem of key storage is also present in the handling of removable media. Since an entire volume (tape or disk pack) can be encrypted with the same key (or small set of keys), the size of the problem is reduced. If archival media are encrypted, then the keys must be kept for a long period in a highly reliable way. One solution to this problem would be to store the keys on the units to which they correspond, perhaps even in several different places to avoid local errors on the medium. The keys would have to be protected, of course; a simple way would be to encrypt them with yet a different "master" key. The protection of this master key is absolutely essential to the system's security.

In addition, it is valuable for the access control decision to be dependent on the value of the data being protected, or even on the value of other, related data; salary fields are perhaps the most quoted example. In this case, the software involved, be it applications or system procedures, must maintain its own key table storage in order to examine the cleartext form of the data successfully. That storage, as well as the routines which directly access it, requires a high-quality protection mechanism beyond encryption.

Since a separate, reliable protection mechanism seems required for the heart of a multiuser system, it is not clear that the use of encryption (which requires the implementation of a second mechanism) is advisable for protection within the system. The system's protection mechanism can usually be straightforwardly extended to provide all necessary protection facilities.

2. SYSTEM AUTHENTICATION

Authentication refers to the identification of one member of a communication to the other in a reliable, unforgeable way. In early interactive computer systems, the primary issue was to provide a method by which the operating system could determine the identity of the user who was attempting to log

in. Typically, user identification involves supplying confidential parameters, such as passwords or answers to personal questions. There was rarely any concern over the machine identifying itself to the user.

In networks, however, mutual authentication is of interest: Each "end" of the channel may wish to assure itself of the identity of the other end. Quick inspection of the class of methods used in centralized systems shows that a straightforward extension is unacceptable. Suppose each participant must send a secret password to the other. Then the first member that sends the password is exposed. The other member may be an imposter, who has now received the necessary information in order to pose to other nodes as the first member. Extension to a series of exchanges of secret information will not solve the problem; it only forces the imposter into a multistep procedure.

There are a number of straightforward encryption-based authentication protocols which provide reliable mutual authentication without exposing either participant. The methods are robust in the face of all the network security threats mentioned earlier. The general principle involves the encryption of a rapidly changing unique value using a prearranged key and has been independently rediscovered by a number of people [FEIS75, KENT76, POPE78]. An obvious application for such protocols is to establish a mutually agreed upon sequence number or block chaining initial value that can be used to authenticate communications over a secure channel whose keys have been used before. The sequence number or value should either be one that has not been used before, or it should be selected at random, in order to protect against undetected replay of previous messages.

Here is an outline of a simple, general authentication sequence between nodes A and B. At the end of the sequence A has reliably identified itself to B. A similar sequence is needed for B to identify itself to A. Typically, one expects to interleave the messages of both authentication sequences.

Assume that in the authentication sequence A uses a secret key associated with itself. The reliability of the authentication depends only on the security of that key. Assume that B holds A's matching key (as well as the matching keys for all other parties to which B might talk).

1) B sends A, in cleartext, a random, unique data item, in this case the current time of day as known to B.
2) A encrypts the received time of day using its authentication key and sends the resulting ciphertext to B.
3) B decrypts A's authentication message, using A's matched key, and compares it with the time of day which B had sent. If they match, then B is satisfied that A was the originator of the message.

This simple protocol exposes neither A nor B if the encryption algorithm is strong, since it should not be possible for a cryptanalyst to be able to deduce the key from the encrypted time of day. This is true even if the cryptanalyst knows the corresponding cleartext time of day. Further, since the authentication messages change rapidly, recording an old message and retransmitting is not effective.

To use such an authentication protocol to establish a sequence number or initial value for block chaining, A includes that information, before encryption, in its step 2 message to B.

3. KEY MANAGEMENT

For several participants in a network conversation to communicate securely, it is necessary for them to obtain matching keys to encrypt and decrypt the transmitted data. It should be noted that a matched pair of keys forms a logical channel which is independent of all other such logical channels but as real as any channel created by a network's transmission protocols. Possession of the key admits one to the channel. Without the key the channel is unavailable. Since the common carrier function of the network is to provide many communication channels, how the keys which create the corresponding necessary private channels are supplied is obviously an important matter. The following sections describe various key distribution methods for both conventional- and public-key encryption systems.

3.1 Conventional-Key Distribution

As there are, by assumption, no suitable transmission media for the keys other than the physical network, it is necessary to devise means to distribute keys over the same physical channels by which actual data are transmitted. The safety of the logical channels over which the keys are to pass is crucial. Unfortunately, the only available method by which any data, including the keys, can be transmitted in a secure manner is through the very encryption whose initialization is at issue. This seeming circularity is actually easily broken through limited prior distribution of a small number of keys by secure means. The usual approach involves designating a host machine or a set of machines [HELL78] on the network to play the role of key distribution center (KDC), at least for the desired connection. It is assumed that a pair of matched keys has been arranged previously between the KDC and each of the potential participants, say $A_1, A_2, \ldots, A_m$. One of the participants, A_i, sends a short message to the KDC asking that matched key pairs be distributed to all the A's, including A_i. If the KDC's protection policy permits the connection, secure messages containing the key and other status information will be sent to each A over the prearranged channels. Data can then be sent over the newly established logical channel. The prearranged key distribution channels carry a low quantity of traffic, and thus, recalling the discussion in Section 1, the keys can be changed relatively infrequently by other means.

This general approach has many variations to support properties such as a distributed protection policy, integrity in the face of crashes, and the like. Some of these are discussed below.

3.1.1 *Centralized Key Control*

Perhaps the simplest form of the key distribution method employs a single KDC for the entire network. Therefore n prearranged matched key pairs are required for a network with n distinguishable entities. An obvious disadvantage of this unadorned approach is its effect on network reliability. If communication with the KDC becomes impossible, either because the node on which the KDC is located is down or because the network itself breaks, then the establishment of any further secure communication channels is impossible. If the overall system has been constructed to prevent any interuser communication in other than a secure manner, then the entire network eventually stops. This design for distributed systems is, in general, unacceptable except when the underlying communications topology is a star and the KDC is located at the center. Note, however, that this drawback can be fairly easily remedied by the availability of redundant KDCs in case of failure of the main facility.[2] The redundant facility can be located at any site which supports a secure operating system and provides appropriate key generation facilities. Centralized key control can quite easily become a performance bottleneck, however.

Needham and Schroeder present an example of how such a KDC would operate [NEED78]. Assume that A and B each have a secret key, K_a and K_b, known only to themselves and the KDC. To establish a connection, A sends a request to the KDC requesting a connection to B and includes an identifier (a random number perhaps). The KDC will send back to A: i) a new key K_c to use in the connection, ii) the identifier, iii) a copy of the request, and iv) some information which A can send to B to establish the connection and prove A's identity. That message from the KDC to A is encrypted with A's secret key K_a. Thus, A is the only one who can receive it, and A knows that it is genuine. In addition, A can check the identifier to verify that it is not a replay of some previous request, and can verify that his original cleartext message was not altered before reception by the KDC.

---

[2] The redundant KDCs form a simple distributed, replicated database, where the replicated information includes private keys and permission controls. However, the database is rarely updated, and when updated, there are no serious requirements for synchronization among the updates It is not necessary for copies of a key at all sites to be updated simultaneously, for example Therefore, little additional complexity from the distributed character of the key management function would be expected.

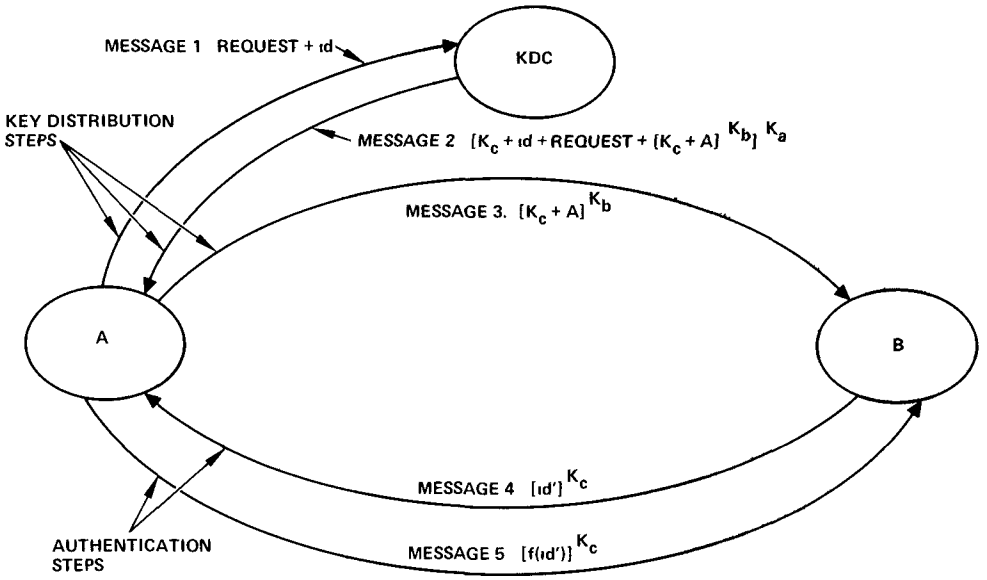

FIGURE 1. Key distribution and conversation establishment: conventional key algorithms. Note: [i'] denotes the cryptogram obtained from the cleartext i, encrypted with key j.

Once A has received this message, A sends to B the data from the KDC intended for B. Those data include the connection key K_c, as well as A's identity, all encrypted by B's secret key. Thus B now knows the new key, that A is the other party, and that all this came from the KDC. However B does not know that the message he just received is not a replay of some previous message. Thus B must send an identifier to A encrypted by the connection key, upon which A can perform some function and return the result to B. Now B knows that A is current, i.e., there has not been a replay of previous messages. Figure 1 illustrates the messages involved. Of the five messages, two can be avoided, in general, by storing frequently used keys at the local sites, a technique known as caching.

3 1.2 *Fully Distributed Key Control*

Here it is possible for every "intelligent" node in the network to serve as a KDC for certain connections. (We assume some nodes are "dumb," such as terminals or possibly personal computers.) If the intended participants $A_1, A_2, \ldots, A_m$ reside at nodes $N_1, N_2, \ldots, N_m$, then only the KDCs at each of those nodes need be involved in the protection decision. One node chooses the key, and sends messages to each of the other KDCs. Each KDC can then decide whether the attempted channel is to be permitted and reply to the originating KDC. At that point the keys would be distributed to the participants. This approach has the obvious advantage that the only nodes which must be properly functioning are those which support the intended participants. Each of the KDCs must be able to communicate with all other KDCs in a secure manner, implying that $n^*(n-1)/2$ matched key pairs must have been arranged. Of course, each node needs to store only $n-1$ of them. For such a method to be successful, it is also necessary for each KDC to communicate with the participants at its own node in a secure fashion. This approach permits each host to enforce its own security policy if user software is forced by the local system architecture to use the network only through encrypted channels. This arrangement has appeal in decentralized organizations.

3.1 3 *Hierarchical Key Control*

This method distributes the key control function among "local," "regional," and

"global" controllers. A local controller is able to communicate securely with entities in its immediate logical locale, that is, with those nodes for which matched key pairs have been arranged. If all the participants in a channel are within the same region, then the connection procedure is the same as for centralized control. If the participants belong to different regions, then it is necessary for the local controller of the originating participant to send a secure message to its regional controller, using a prearranged channel. The regional controller forwards the message to the appropriate local controller, who can communicate with the desired participant. Any of the three levels of KDCs can select the keys. The details of the protocol can vary at this point, depending on the exact manner in which the matched keys are distributed. This design approach obviously generalizes to multiple levels in the case of very large networks. It is analogous to national telephone exchanges, where the exchanges play a role very similar to the KDCs.

One of the desirable properties of this design is the limit it places on the combinatorics of key control. Each local KDC only has to prearrange channels for the potential participants in its area. Regional controllers only have to be able to communicate securely with local controllers. While the combinatorics of key control may not appear difficult enough to warrant this kind of solution, in the subsequent section on levels of integration we point out circumstances in which the problem may be very serious.

The design also has a property not present in either of the preceding key control architectures: local consequences of local failures. If any component of the distributed key control facility should fail or be subverted, then only users local to the failed component are affected. Since the regional and global controllers are of considerable importance to the architecture, it would be advisable to replicate them so that the crash of a single node will not segment the network.

All of these key control methods permit easy extension to the interconnection of different networks, with differing encryption disciplines. The usual way to connect different networks, which often employ different transmission protocols, is to have a single host called a gateway common to both networks [CERF78, BOGG80]. Internetwork data are sent to the gateway, which forwards them toward the final destination. The gateway is responsible for any format conversions, as well as for the support of both systems' protocols and naming methods. If the networks' transmissions are encrypted in a manner similar to that described here, then the gateway might be responsible for decrypting the message and reencrypting it for retransmission in the next network. This step is necessary if the encryption algorithms differ, or if there are significant differences in protocol. If the facilities are compatible, then the gateway can merely serve as a regional key controller for both networks, or even be totally uninvolved.

There are strong similarities among these various methods of key distribution, and differences can be reduced further by designing hybrids to gain some of the advantages of each. Centralized control is a degenerate case of hierarchical control. Fully distributed control can be viewed as a variant of hierarchical control. Each host's KDC acts as a local key controller for that host's entities and communicates with other local key controllers to establish connections. In that case, of course, the communication is direct, without a regional controller required.

3.2 Public-Key-Based Distribution Algorithms

The public-key algorithms discussed earlier have been suggested as candidates for key distribution methods that might be simpler than those described in the preceding sections. Recall that K', the key used to decipher the encoded message, cannot be derived from K, the key used for encryption, or from matched encrypted and cleartext. Therefore, each user A, after obtaining a matched key pair $\langle K, K' \rangle$, can publicize his key K. Another user B, wishing to send a message to A, can employ the publicly available key K. To reply, A employs B's public key. At first glance this mechanism

seems to provide a simplified way to establish secure communication channels. No secure dialogue with a key controller to initiate a channel appears necessary.

The idea is that an automated "telephone book" of public keys could be made available. Whenever user A wishes to communicate with user B, A merely looks up B's public key in the book, encrypts the message with that key, and sends it to B [DIFF76b]. There is no key distribution problem here at all. Further, no central authority is required to set up the channel between A and B.

This idea, however, is incorrect: Some form of central authority is needed, and the protocol involved is no simpler nor any more efficient than one based on conventional algorithms [NEED78]. First, the safety of the public-key scheme depends critically on the correct public key being selected by the sender. If the key listed with a number in the "telephone book" is the wrong one, then the protection supplied by public-key encryption has been lost. Furthermore, maintenance of the (by necessity, machine-supported) book is nontrivial because keys will change, either because of the desire to replace a key which has been used for high amounts of data transmission or because a key has been compromised through a variety of ways. There must be some source of carefully maintained "books" with the responsibility of carefully authenticating any changes and correctly sending out public keys (or entire copies of the book) upon request.

A modified version of Needham and Schroeder's proposal follows. Assume that A and B each have a public key known to the authority and a private key known only to themselves. Additionally, assume the authority has a public key known to all and a private key known only to the authority.

A begins by sending to the authority a time-stamped message requesting communication with B. The authority sends A the public key of B, a copy of the original request, and the time stamp, encrypted using the private key of the authority. A can decrypt this message using the public key of the authority and is thus also sure of the source of the message. The time stamp guarantees that this is not an old message from the authority containing a key other than B's current public key, and the copy of the request permits A to verify that his original cleartext message was not altered.[3]

A can now send messages to B because he knows B's public key. However, to identify himself, as well as to prevent a replay of previous transmissions, A now sends his name and an identifier to B, encrypted in B's public key. B now performs the first two steps above with the authority to retrieve A's public key. Then B sends to A the identifier just received, and an additional identifier, both encrypted with A's public key. A can decrypt that message and is now sure that he is talking to the current B. A must now send back the new identifier to B so that B can be sure he is talking to a current A. These messages are displayed in Figure 2. The above protocol contains seven messages, but four of them, those which retrieve the public keys, can be largely dispensed with by local caching of public keys. Thus, as in the conventional-key distribution example, we again find three messages are needed.

Some public-key advocates have suggested ways other than caching in order to avoid requesting the public key from the central authority for each communication. One such proposal is the use of certificates [KOHN78]. A user can request that his public key be sent to him as a *certificate*, which is a user/public-key pair, together with some certifying information. For example, the user/public-key pair may be stored as a signed message[4] from the central authority. When the user wishes to communicate with other users, he sends the certificate to them. They each can check the validity of the certificate, using the certifying information, and then retrieve the public key. Thus the central authority is needed only once, when the initial certificate is requested.

Both certificates and caching have several problems. First, the mechanism used to store the cache of keys must be correct.

---

[3] These initial steps are essentially an adaptation of the authentication protocol given in Section 2.

[4] See Section 9 for a discussion of digital signatures.

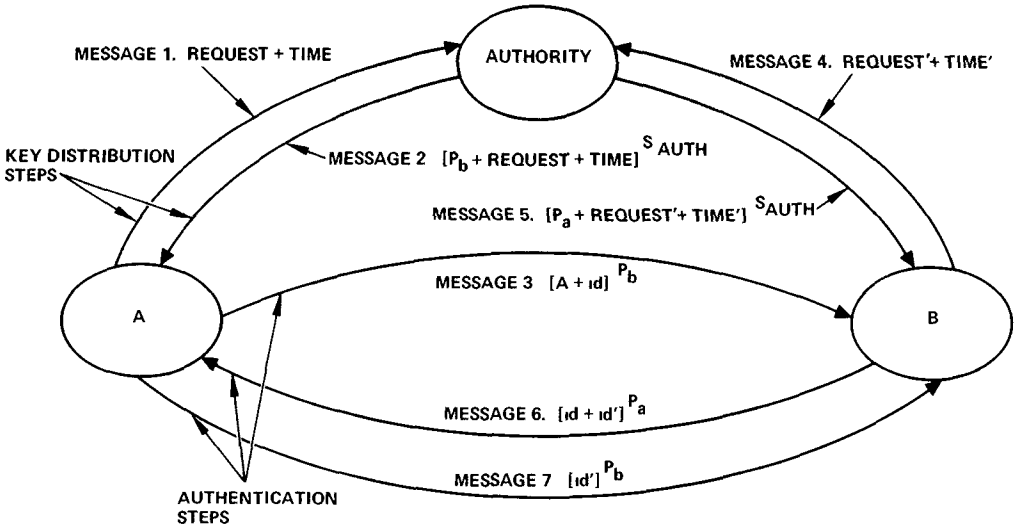

FIGURE 2. Key distribution and conversation establishment: public-key algorithms. Note P_i is public key for i, S_i is secret key for i.

Second, the user of the certificate must decode it and check it (verify the signature) each time before using it, and he must also have a secure and correct way of storing the key. Perhaps most important, as keys change, the cache and old certificates become obsolete. This is essentially the capability revocation problem revisited [REDE74]. Either the keys must be verified (or re-requested) periodically, or a global search must be made whenever invalidating a key. Notice that even with the cache or certificates, an internal authentication mechanism is still required.

Public-key systems also have the problem that it is more difficult to provide protection policy checks. In particular, conventional encryption mechanisms easily allow protection policy issues to be merged with key distribution. If two users may not communicate, then the key controller can refuse to distribute keys.[5] However, public-key systems imply the knowledge of the public keys. Methods to add protection checks to public-key systems add an additional layer of mechanism.

3.3. Comparison of Public- and Conventional-Key Distribution for Private Communication

It should be clear that both of the above protocols establish a secure channel, and that both require the same amount of overhead to establish a connection (three messages). Even if that amount had been different by a message or two, the overhead is still small compared to the number of messages for which a typical connection will be used.

The above protocols can be modified to handle multiple authorities; such modifications have also been performed by Needham and Schroeder [NEED78]. Again, the number of messages can be reduced to three by caching.

It should also be noted that the safety of these methods depends only on the safety of the secret keys in the conventional method or the private keys in the public-key method. Thus an equivalent amount of secure storage is required.

One might suspect, however, that the software required to implement a public-key authority would be simpler than that for a KDC, and therefore it would be easier to certify its correct operation. If this view were correct, it would make public-key-based encryption potentially superior to

[5] This approach blocks communication if the host operating systems are constructed in such a way as to prohibit cleartext communication over the network

conventional algorithms, despite the equivalent protocol requirements. It is true that the contents of the authority need not be protected against unauthorized reference, since the public keys are to be available to all, while the keys used in the authentication protocol between the KDC and the user must be protected against reference. However, the standards of software reliability which need to be imposed on the authority for the sake of correctness are not substantially different from those required for the development of a secure KDC. More convincing, all of the KDC keys could be stored in encrypted form, using a KDC master key, and only decrypted when needed. Then the security of the KDC is reduced to protection of the KDC's master key and of the individual keys when in use. This situation is equivalent to the public-key repository case, since there the private key of the repository must be safely stored and protected during use.

It has also been pointed out that a conventional KDC, since it issued the conversation key, can listen in and in fact generate what appear to be valid messages. Such action cannot be done by the public-key repository. This distinction is minor however. Given that both systems require a trusted agent, it is a simple matter to add a few lines of certified correct code to the conventional-key agent (the KDC) that destroys conversation keys immediately after distribution. Thus the system characteristics of both conventional- and public-key algorithms, as used to support private communication, are more similar than initially expected.

4. LEVELS OF INTEGRATION

There are many possible choices of endpoints for the encryption channel in a computer network, each with its own trade-offs. In a packet-switched network, one could encrypt each line between two switches separately from all other lines. This is a low-level choice and is often called *link encryption.* Instead, the endpoints of the encryption channels could be chosen at a higher architectural level—at the host machines which are connected to the network. Thus the encryption system would support host-host channels, and a message would be encrypted only once as it was sent through the network (or networks) rather than being decrypted and reencrypted a number of times, as implied by the low-level choice. In fact, one could choose an even higher architectural level: Endpoints could be individual processes within the operating systems of the machines that are attached to the network. If the user were employing an intelligent terminal, then the terminal would be a candidate for an endpoint. This viewpoint envisions a single encryption channel from the user directly to the program with which he is interacting, even though that program might be running on a site other than the one to which the terminal is connected. This high-level choice of endpoints is sometimes called *end-to-end encryption.*

The choice of architectural level in which the encryption is to be integrated has many ramifications. One of the most important is the combinatorics of key control versus the amount of trusted software.

In general, as one considers higher and higher system levels, the number of identifiable and separately protected entities in the system tends to increase, sometimes dramatically. For example, while there are less than a hundred hosts attached to the Arpanet [ROBE73], at a higher level there often are over a thousand processes concurrently operating, each one separately protected and controlled. The number of terminals is of course also high. This numerical increase means that the number of previously arranged secure channels—that is, the number of separately distributed matched key pairs—is correspondingly larger. Also, the rate at which keys must be generated and distributed can be dramatically increased.

In return for the additional cost and complexity which result from higher level choices, there can be significant reduction in the amount of software whose correct functioning has to be ensured. This issue is very important and must be carefully considered. It arises in the following way. When the lowest level (i.e., link encryption) is chosen, the data being communicated exist in cleartext form as they are passed by the switch from one encrypted link to the next. Therefore the software in the switch must be trusted not to intermix packets of differ-

ent channels. If a higher level is selected, then protection errors in the switches are of little consequence. If the higher level chosen is host to host, however, operating system failures are still serious, because the data exist as cleartext while they are system resident.

In principle then, the highest level integration of encryption is most secure. However, it is still the case that the data must be maintained in cleartext form in the machine upon which processing is done. The more classical methods of protection within individual machines are still necessary, and the value of very high level end-end encryption is thereby somewhat lessened. A rather appealing choice of level that integrates effectively with kernel-structured operating system architectures is outlined in the case study in Section 7.

Another operational drawback to high-level encryption should be pointed out. Once the data are encrypted, it is difficult to perform meaningful operations on them. Many front end systems provide such low-level functions as packing, character erasures, and transmission on end-of-line or control-character detect. If the data are encrypted when they reach the front end, then these functions cannot be performed. Any channel processing must be done above the level at which encryption takes place, despite the fact that performance and considerations such as the above sometimes imply a lower level.

5. ENCRYPTION PROTOCOLS

Network communication protocols concern the discipline imposed on messages sent throughout the network to control virtually all aspects of data traffic, both in amount and direction. Choice of protocol has dramatic impacts on the flexibility and bandwidth provided by the network. Since encryption facilities provide a potentially large set of logical channels, the encryption protocols by which the operation of these channels is managed also has significant impact on system architecture and performance.

There are several important questions which any encryption protocol must answer:

1) How is the initial cleartext/ciphertext/cleartext channel from sender to receiver and back established?
2) How are cleartext addresses passed by the sender around the encryption facilities to the network without providing a path by which cleartext data can be inadvertently or intentionally leaked by the same means?
3) What facilities are provided for error recovery and resynchronization of the protocol?
4) How are channels closed?
5) How do the encryption protocols interact with the rest of the network protocols?
6) How much software is needed to implement the encryption protocols? Does the security of the network depend on this software?

One wishes a protocol which permits channels to be dynamically opened and closed, allows the traffic flow rate to be controlled (by the receiver presumably), and provides reasonable error handling, all with a minimum of mechanism upon which the security of the network depends. The more software involved, the more one must be concerned about the safety of the overall network. Performance resulting from use of the protocol must compare favorably with the attainable performance of the network using other protocols not including encryption. One would prefer a general protocol which could also be added to the existing networks, disturbing their existing transmission mechanisms as little as possible. The appropriate level of integration of encryption or the method of key distribution must be considered as well.

Fortunately, the encryption channel can be managed independently of the conventional communication channel, which is responsible for communication initiation and closing, flow control, error handling, and the like. As a result, many protocol questions can be ignored by the encryption facilities and can be handled by conventional means.

In Section 7 we outline a complete protocol in order to illustrate the ways in which these considerations interact and the independence that exists. The case considered

employs distributed key management and an end-to-end architecture, all added to an existing network.

6. CONFINEMENT

To confine a program, process, or user means to render it unable to communicate other than through the explicitly controlled paths. Often improper communications are possible through subtle, sometimes timing-dependent, channels. As an example, two processes might bypass the controlled channels by affecting each other's data throughput. Although many such improper channels are inherently error prone, the users may employ error detection and correction protocols to overcome that problem.

Unfortunately, the confinement problem in computer networks is particularly difficult to solve because most network designs require some information to be transmitted in cleartext form. This cleartext information, although limited, can be used for the passage of unauthorized information. In particular, the function of routing a message from computer to computer toward its final destination requires that the headers which contain network addresses and control information be in cleartext form, at least inside the switching centers. A malicious user, cooperating with a penetrator, can send data by the ordering of messages between two communication channels. Even though the data of the communications are encrypted, the headers often are transmitted in cleartext form, unless link encryption is also used to encrypt the entire packet, including header. In any case, the routing task, often handled in large networks by a set of dedicated interconnected machines which form a subnet, requires host addresses in the clear within the switching machines. Thus a penetrator who can capture parts of the subnetwork can receive information. The only solutions to this problem appear to be certification of the secure nature of some parts of the subnetwork and host hardware/software. Work is in progress at the University of Texas on the application of program verification methods to this problem [GOOD77].

Certain confinement problems remain even if certification is applied as suggested. For example, the protocol-implementing software in a given system usually manipulates communications for several users simultaneously. Either this software must be trusted, or data must be encrypted before it reaches this software. Even in this latter case, certain information may be passed between the user and the network software, and thus, potentially, to an unauthorized user. As an example, if a queue is used to hold information waiting to be sent from the user to the network, the user can receive information by noting the amount drained from this queue by the network software. In almost any reasonable implementation on a system with finite resources, the user will at least be able to sense the time of data removal, if not the amount.

How well current program verification and certification methods apply here is open to question, since these confinement channels are quite likely to exist even in a correct implementation. That is, any feasible design seems to include such channels.

Given the difficulty of confinement enforcement, it is fortunate that most applications do not require it.

7. NETWORK ENCRYPTION PROTOCOL CASE STUDY: PRIVATE COMMUNICATION AT PROCESS-PROCESS LEVEL

It is useful to review a case study of how encryption was integrated into a real system in order to recognize the importance of the various issues already presented. The example here was designed and implemented for the Arpanet, and is described in more detail by Popek and Kline [POPE78]; here we only outline the solution in general terms. The goal is to provide secure communication that does not involve application software in the security facilities. We also wish to minimize the amount of trusted system software.

The protocol provides process-to-process channels and guarantees that it is not possible for application software running within the process to cause cleartext to be transmitted onto the network. Basic operation of the protocol is suggested in Figure 3. It is assumed, in keeping with the discussion in Section 1.6, that the system software

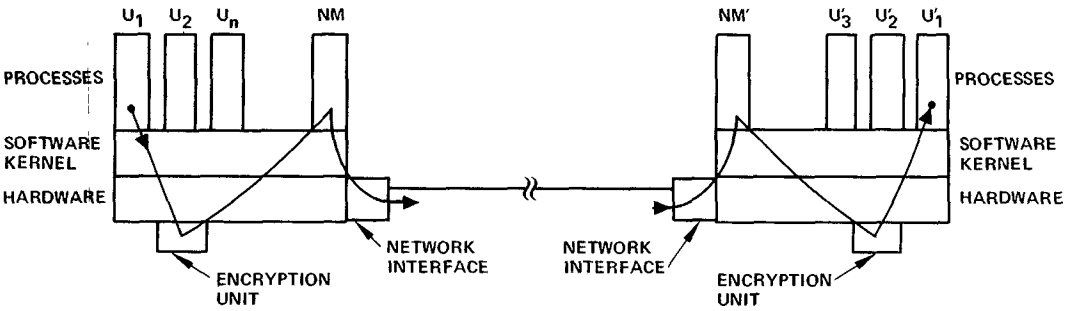

FIGURE 3. Data flow in process-to-process encrypted channels.

base at each node is a suitably small, secure operating system kernel, which operates correctly.

It is also expected that the amount of software involved in management of the network from the operating system's point of view is substantial; therefore one does not wish to trust its correct operation.[6] Responsibilities of that software include establishing communications channels, supporting retransmission when errors are detected, controlling data flow rates, multiplexing multiple logical channels on the (usually) single physical network connection, and assisting or making routing decisions. We call the modules which provide these functions the network manager.

Let us assume for the moment that the keys have already been distributed and logical channels established so far as the network managers are concerned. The operating system nucleus in each case has been augmented with new calls: *Encrypt*(*channel name, data*) and *Decrypt*(*channel name, data destination*). Whenever a process wishes to send an encrypted block of data, it issues the Encrypt call. The nucleus takes the data, causes them to be encrypted, and informs the network manager, which can read the block into its workspace. If we assume that the network manager knows what destination site is intended (which it must learn as part of establishing the logical channel), it then can place a cleartext header on the encrypted block and send it out onto the network. The cleartext header is essential so that switching computers which typically make up a network can route the block appropriately.[7]

When the block arrives at the destination host computer, the network manager there reads it in and strips off the header. It then tells the kernel the process for which the block is intended. The kernel informs the process, which can issue a Decrypt call, causing the data to be decrypted with the key previously arranged for that process. If this block really is intended for this process (i.e., encrypted with the matching key), then the data are successfully received. Otherwise, decryption with the wrong key yields nonsense. The encrypt and decrypt functions manage sequence numbers in a manner invisible to the user, as discussed in Section 1.3.

Clearly this whole mechanism depends on suitable distribution of keys, together with informing the network managers in a coordinated way of the appropriate endpoints of the channel. It is worth noting at this stage that matched keys form a well-defined communication channel, and that in the structure just outlined, it is not possible for processes to communicate to the network or the network manager directly; only the encrypt and decrypt functions can be used for this purpose. It is for this latter reason that application software cannot communicate in cleartext over the network, an advantage if that code is not trusted (the usual assumption in military examples).

[6] As an example, in the Arpanet software for the Unix operating system, the network software is comparable in size to the operating system itself

[7] Network encryption facilities must, in general, provide some way to supply the header of a message in cleartext, even though the body is encrypted. Otherwise every node on possibly multiple networks has to be able to examine every message, this is not practical.

7.1 Initial Connection

To establish the secure channel, several steps are necessary. The local network manager must be told with whom the local process wishes to communicate. This would be done by some highly constrained means. The network manager must communicate with the foreign network manager and establish a name for this channel, as well as other state information such as flow control parameters. The network manager software involved need not be trusted. Once these steps are done, encryption keys need to be set up in a safe way.

We first outline how this step would be carried out employing conventional encryption with fully distributed key management; then we comment on how it would change if public-key systems were used.

Assume that there is a kernel-maintained key table which has entries of the form:

foreign host name,
channel name,
sequence number,
local process name,
key.

There are also two additional kernel calls. *Open(foreign process name, local process name, channel name, policy-data)* makes the appropriate entry in the key table (if one is not already there for the given channel), setting the sequence number to an initial value and sending a message to the foreign kernel of the form ⟨local process name, channel name, policy-data, key⟩.[8]

If there already is an entry in the local key table, it should have been caused by the other host's kernel. In that case Open checks to make sure that the sequence number has been initialized and does not generate a key—rather it sends out the same message, less the key. *Close(channel name)* deletes the indicated entry in the local key table, and sends a message to the foreign kernel to do the same.

The policy-data supplied in the Open call, such as classification/clearance information, will be sent to the other site involved in the channel so that it too will have the relevant basis for deciding whether or not to allow this channel to be established.

Once both sides have issued corresponding Open calls, the process can communicate. The following steps illustrate the overall sequence in more detail. The host machines involved are numbered 1 and 2. Process A is at host 1 and B is at host 2. The channel name will be x. The notation NM @ ι denotes "network manager at site i."

1) A informs NM @ 1 "connect using x to B @ 2." This message can be sent locally in the clear. If confinement between the network manager and local processes is important, other methods can be employed to limit the bandwidth between A and NM.
2) NM @ 1 sends control messages to NM @ 2, including whatever host machine protocol messages are required.[9]
3) NM @ 2 receives an interrupt indicating normal message arrival, performs an I/O call to retrieve it, examines the header, determines that it is the recipient, and processes the message.
4) NM @ 2 initiates step 2 at site 2, leading to step 3's being executed at site 1 in response. This exchange continues until NM @ 1 and NM @ 2 establish a logical channel, using x as their internal name for the channel.
5) NM @ 1 executes Open(B, A, x, policy-data).
6) In executing the Open, the kernel @ 1 generates or obtains a key, makes an entry in its key table, and sends a message over its secure channel to the kernel @ 2, which in turn makes a corresponding entry in its table and interrupts NM @ 2, giving it the triple ⟨B, A, x⟩.
7) NM @ 2 issues the corresponding Open(A, B, x, policy-data′). This call interrupts B and eventually causes the appropriate entry to be made in the kernel table at host 1. The making of that entry interrupts NM @ 1 and A @ 1.

[8] The reader will note that the kernel-to-kernel message generated by the Open call must be sent securely and therefore must employ a previously arranged key The network manager must also be involved, since only it contains the software needed to manage the network

[9] The host-host protocol messages would normally be sent encrypted using the NM-NM key in most implementations.

8) A and B can now use the channel by issuing successive Encrypt and Decrypt calls.

There are a number of places in the mechanisms just described where failure can occur. If the network software in either of the hosts fails or decides not to open the channel, no kernel calls are involved and standard protocols operate. (If user notification is permitted, an additional confinement channel is present.) An Open may fail because the name x supplied was already in use, a protection policy check was not successful, or the kernel table was full. The caller is notified. He may try again. In the case of failure of an Open, it may be necessary for the kernel to execute most of the actions of Close to avoid race conditions that can result from other methods of indicating failure to the foreign site.

The encryption mechanism just outlined contains no error correction facilities. If messages are lost, or sequence numbers are out of order or duplicated, the kernel merely notifies the user and network software of the error and renders the channel unusable.[10] This action is taken on all channels, including the host-host protocol channels as well as the kernel-kernel channels. For every case but the last, Close calls must be issued and a new channel created via Opens. In the last case, the procedures for bringing up the network must be used.

This simple-minded view is acceptable in part because the error rate which the logical encryption channel sees can be quite low. That is, the encryption channel is built on top of lower level facilities supplied by conventional network protocols, some implemented by the NM, which can handle transmission errors (forcing retransmission of errant blocks, for example) before they are visible to the encryption facilities. On highly error prone channels, additional protocol at the encryption level may still be necessary. See KENT76 for a discussion of resynchronization of the sequencing supported by the encryption channel.

From the protection viewpoint, one can consider the collection of NMs across the network as forming a single (distributed) domain. They may exchange information freely among themselves. No user process can send or receive data directly to or from an NM, except via narrow bandwidth channels through which control information is sent to the NM and status and error information is returned. These channels can be limited by adding parameterized calls to the kernel to pass the minimum amount of data to the NMs and having the kernel post, to the extent possible, status reports directly to the processes involved. The channel bandwidth cannot be zero, however.

The protocols in this case study can also be modified to use public-key algorithms. The kernel, upon receiving the Open request, should retrieve the public key of the recipient. Presumably, the kernel would employ a protocol with the authority to retrieve the public key and then utilize the authentication mechanisms described in the protocols of Section 2.

More precisely, in step 6 above, when the kernel receives the Open call, it would retrieve the public key, either by looking it up in a cache or requesting it from the central authority, or via other methods such as certificates. Once the key is retrieved, the kernel would send a message to the other kernel over the secure kernel-kernel channel, identifying the user and supplying those policy and authentication parameters required. The other kernel, upon receipt of that message, would retrieve the user's private key (from wherever local user private keys are stored) and continue the authentication sequence.

7.2 System Initialization Procedures

The task of initializing the network software is composed of two important parts. First, it is necessary to establish keys for the secure kernel-kernel channels and the NM-NM channels. Next, the NM can initialize itself and its communications with

[10] Recall that these sequence numbers are added to the cleartext by the kernel Encrypt call before encryption. They are removed and checked after decryption by a Decrypt call issued at the receiving site before delivery to the user. Hence, if desired, sequence numbers can be handled by the encryption unit itself and never be seen by kernel software. If such a choice is made, then the conventional network protocols supported by the NM will need another set of sequence numbers for error control.

other NMs. Finally, the kernel can initialize its communications with other kernels. This latter problem is essentially one of mutual authentication of each kernel with the other member of the pair, and appropriate solutions depend on the expected threats against which protection is desired.

The initialization of the kernel-kernel channel and the NM-NM channel key table entries requires that the kernel maintain initial keys for this purpose. The kernel cannot obtain these keys using the above mechanisms at initialization because they require the prior existence of the NM-NM and kernel-kernel channels. Thus this circularity requires the kernel to maintain at least two key pairs.[11] However, such keys could be kept in read-only memory of the encryption unit if desired.

The initialization of the NM-NM communications then proceeds as it would if encryption were not present. Once this NM-NM initialization is complete, the kernel-kernel connections could be established by the NM. At this point, the system would be ready for new connection establishment. It should be noted that if desired, the kernels could then set up new keys for the kernel-kernel and NM-NM channels, thus using the initialization keys for a short time only. To avoid overhead at initialization time and to limit the sizes of kernel key tables, NMs probably should only establish channels with other NMs when a user wants to connect to that particular foreign site, and perhaps the NM-NM channel should be closed after all user channels are closed.

This case study should serve to illustrate many of the issues present in the design of a suitable network encryption facility.

7.3. Symmetry

The case study portrays a basically symmetric protocol suitable for use by intelligent nodes, a fairly general case. However, in some instances one of the pair lacks algorithmic capacity, as illustrated by simple hardware terminals or simple microprocessors. Then a strongly asymmetric protocol is required, where the burden of establishing secure communications falls on the more powerful of the pair.

A form of this problem might also occur if encryption is not handled by the system, but by the user processes themselves. Then for certain operations, such as sending mail, the receiving user process might not even be present. (Note that such an approach may not guarantee the encryption of all network traffic.) The procedures outlined in the next section are oriented toward reducing the work of one of the members of the communicating pair.

8. NETWORK MAIL

Recall that network mail may often be short messages, to be delivered as soon as possible to the recipient site and stored there, even if the intended receiver is not currently logged in.

Assume that a user at one site wishes to send a message to a user at another site, but because the second user may not be signed on at the time, a system process (sometimes called a "daemon") is used to receive the mail and deliver it to the user's "mailbox" file for his later inspection. It is desirable that the daemon process not require access to the cleartext form of the mail, for that would require trusting the mail receiver mechanism. This task can be accomplished by sending the mail to the daemon process in encrypted form and having the daemon put that encrypted data directly into the mailbox file. The user can decrypt the data when he signs on to read his mail.

In either the conventional- or public-key case, the protocols described in Section 3 can be employed with only slight modifications. In the conventional-key case, the last two messages, those which exchange an identifier to ensure that the channel is current, must be dropped (since the recipient may not be present). After the sender requests and gets a key K and a copy of K encrypted with the receiver's secret key, he appends the encrypted mail to the encrypted K and sends both to the receiver.

[11] In a centralized key controller version, the only keys needed would be those for the channel between the key controller's NM and the host's NM, and for the channel between the key controller's kernel and the host's kernel. In a distributed key management system, keys would be needed for each key manager

The receiving mail daemon can deliver the mail and key (both still encrypted), and the intended recipient can decrypt and read it at his leisure.

In the case of public keys, the sender retrieves the recipient's public key via an exchange with the repository, encrypts the mail, and sends it to the receiving site. Again the mail daemon delivers the encrypted mail, which can be read later by the recipient since he knows his private key. Again, the authentication part of the public-key protocol must be dropped. In both of these approaches, since the authentication steps were not performed, the received mail may be a replay of a previous message. If detecting duplicate mail is important, the receiver must keep records of previous mail.

Both mechanisms outlined above do guarantee that only the desired recipient of a message will be able to read it. However, as pointed out, the recipient is not guaranteed the identity of the sender. This problem is essentially that of digital signatures, which is discussed in the next section.

9. DIGITAL SIGNATURES

Applications such as bank transactions, military command and control orders, and contract negotiations, will require digital signatures. At first, it appeared that public-key methods would be superior to conventional ones for use in digital message signatures. The method, assuming a suitable public-key algorithm, is for the sender to encode the mail with his *private* key and then send it. The receiver decodes the message with the sender's *public* key. The usual view is that this procedure does not require a central authority, except to adjudicate an authorship challenge. However, two points should be noted. First, a central authority is needed by the recipient for aid in deciphering the first message received from any given author (to retrieve the corresponding public key, as mentioned in Section 3.2). Second, the central authority must keep all old values of public keys in a reliable way to properly adjudicate conflicts over old signatures (consider the relevant lifetime of a signature on a real estate deed, for example).

Furthermore, and more serious, the unadorned public-key-signature protocol just described has an important flaw. The author of signed messages can effectively disavow and repudiate his signatures at any time, merely by causing his secret key to be made public or "compromised" [SALT78]. When such an event occurs, either by accident or intention, all messages previously "signed" using the given private key are invalidated, since the only proof of validity has been destroyed. Because the private key is now known, anyone could have created any message claimed to have been sent by the given author. None of the signatures can be relied upon.

Hence the validity of a signature on a message is only as safe as the *entire* future protection of the private key. Further, the ability to remove the protection resides precisely with the individual (the author) who should not hold that right. That is, one important purpose of a signature is to indicate responsibility for the content of the accompanying message in a way that cannot be later disavowed.

The situation with respect to signatures using conventional algorithms might initially appear slightly better. Rabin [RABI78] proposes a method of digital signatures based on any strong conventional algorithm. Like public-key methods it too requires either a central authority or an explicit agreement between the two parties involved to get matters going.[12] Similarly, an adjudicator is required for challenges. Rabin's method, however, uses a large number of keys, with keys not being reused from message to message. As a result, if a few keys are compromised, other signatures based on other keys are still safe. This is not a real advantage over public-key methods, since one could readily add a layer of protocol over the public-key method to change keys for each message as Rabin does for conventional methods. One could even use a variant of Rabin's scheme itself

[12] In his paper, Rabin describes an initialization method which involves an explicit contract between each pair of parties that wish to communicate with digitally signed messages One can easily instead add a central authority to play this role, using suitable authentication protocols, thus obviating any need for two parties to make specific arrangements prior to exchanging signed correspondence

with public keys, although it is easy to develop a simpler one.

All of the digital signature methods described or suggested above suffer from the problem of repudiation of signature via key compromise. Rabin's protocol or analogs to it merely limit the damage (or, equivalently, provide selectivity!). It appears that the problem is intrinsic to any approach in which the validity of an author's signature depends on secret information which can potentially be revealed, either by the author or other interested parties. Surely improvement would be desirable.

A number of proposals have been made to augment or replace the unadorned approaches just outlined. One, suggested in KLIN79 employs a network-wide distributed signature facility. Others, based on analogs to notaries public in the paper world or replicated, trusted archival facilities, provide a dependable time-stamping mechanism so that authors cannot disavow earlier signed correspondence by causing their keys to be revealed.

9.1 Network-Registry-Based Signatures —A Conventional-Key Approach

The registry solution is based on the obvious approach of interposing some trusted interpretive layer, a secure hardware and/or software "unit," between the author and his signature keys, whatever their form. Then it is a simple matter to organize the collection of units in the network to provide digital signature facilities. Consider all the cooperating units together as a distributed network registry (NR). Some secure communication protocol among the components of the registry is required, but it can be very simple; low-level link-style encryption using conventional encryption would suffice.

Given that such facilities exist, then a simple implementation of digital signatures which does not require specialized protocols or encryption algorithms is as follows:

1) The author authenticates with a local component of the network registry (NR), creates a message, and hands the message to the NR together with the recipient identifier and an indication that a registered signature is desired.
2) The NR (not necessarily the local component) computes a simple characteristic function of the message, author, recipient, and current time; encrypts the result with a key known only to the NR; and forwards the resulting "signature block" to the recipient. The NR only retains the encryption key employed.
3) The recipient, when the message is received, can ask the NR if the message was indeed signed by the claimed author by presenting the signature block and message. Subsequent challenges are handled in the same way.

Certain precautions are needed to ensure the safety of the keys used to encrypt the signature blocks, including the use of different keys between pairs of distributed NR components, and a signature block computation which requires compromise of multiple components before signature validity is affected. For example, several NR components could each generate fragments of the keys being used. There is not even any need for all NR components to be under control of a single centralized authority so long as they can all cooperate.

9.2 Notary-Public- and Archive-Based Solutions

Public-key algorithms can provide safe signature methods also. One straightforward method is based on the behavior of notaries public in the paper world.[13] Briefly, there can be a number of independently operating (but perhaps licensed) *notary public machines* attached to the network. When a signed message has been produced, it can be sent to several of the notary public machines by the author after the author has signed the message himself. The notary public machine time-stamps the message, signs it itself (thereby encoding it a second time), and returns the result to the author. The author can then put the appropriate cleartext information around the doubly signed correspondence and send it to the intended receiver. He checks the notary's signature by decoding with the notary's public key, then decodes the message using

[13] This approach was initially suggested to one of the authors by David Redell.

the author's public key. Several notarized copies can be sent, if desired, to increase safety.

The assumption underlying this method is that most of the notaries can be trusted. Since each notary time-stamps its signature, it is not possible for the original author to disavow prior signed correspondence by "losing" his key at a given time. One might think, however, that it is still possible for someone to claim that his key had been revealed sometime in the past without his knowledge and selective messages forged. This problem can be guarded against by having each notary public return a copy of each notarized message to the author's permanent address. (This "patch" of course raises the question of how notaries are kept reliably informed of permanent addresses.)

Each notary is an independent facility, so that no coordination among notaries is required. Of course, if only one notary exists, then the approach is at best no improvement over the scheme presented in the previous section without multiple NR components. Danger of compromise of the notaries' private keys is reduced by the redundant facilities.

A related way to achieve reliable time registration of signed messages is for there to be a number of independent archival sites where either authors or recipients of signed mail may send copies of correspondence to be time-stamped and stored permanently. Of course, the entire message need not be stored; just a characteristic function will do. Challenges are handled by interrogating the archives. The possibility of an individual's key being compromised and used without his knowledge can be treated in the same way as with notaries public.

9.3 Comparison of Signature Algorithms

The improved conventional-key-based and public-key-based signature algorithms share many common characteristics. They each involve some generally trusted mechanism shared among all those communicating. The safety of signatures still depends on the future protection of keys as before, now including those for the network registry, notaries public, or archive facilities. However, there are several crucial differences from previous protocols. First, the authors of messages do not retain the ability to repudiate signatures at will. Second, the new facilities can be structured so that failure or compromise of several of the components is necessary before signature validity is lost. In the early proposals a single failure could lead to compromise.

10. USER AUTHENTICATION

While digital signatures are important, one must realize that there *must* still exist a guaranteed authentication mechanism by which an individual is authenticated to the system. Any reasonable communication system, of course, ultimately requires such a facility, for if one user can masquerade as another, all signature systems will fail. What is required is some reliable way to identify a user sitting at a terminal—some method stronger than the password schemes used today. Perhaps an unforgeable mechanism based on fingerprints or other personal characteristics will emerge.

11. CONCLUSIONS

This discussion of network security has outlined the issues in developing secure computer networks, as well as presented the context in which encryption algorithms will be increasingly used. It is surprising to note that once the system implications are understood, public-key algorithms and conventional algorithms are largely equivalent.

Indeed, it is highly unlikely that any given class of encryption algorithms will be sufficient alone to provide the various secure functions which will be desired. Master-key/subkey relationships, or k-out-of-n systems[14] are just two examples. Rather than attempt to develop and evaluate the strength of a new encryption system for each such application, it would be preferable to recognize that a *strong extensible system* is necessary. Such a system is one for which new characteristics may be easily added, and where the strength of the addition can be demonstrated in a straightforward, incremental manner. Any strong algorithm, either conventional or public key,

[14] A k-out-of-n system is one in which any k of a set of n keys are sufficient to decrypt, but it is infeasible to do so with any fewer

can serve as the basis for a strong extensible system when combined with additional trusted management algorithms, expressed either in hardware or software. Examples of such mixed systems are given in Section 9. In fact, much of the discussion in this paper suggests that mixed systems are essential. Once that necessity is recognized, pressure to develop encryption algorithms with special characteristics is lessened; instead, more attention is focused on the need for strong algorithms in general.

If one assumes that the purpose of a secure network is mainly to provide private pipes, similar to those supplied by common carriers, then general principles by which secure, common-carrier-based, point-to-point communication can be provided are reasonably well in hand. Of course, in any sophisticated implementation, there will be considerable careful engineering to be done. However, this conclusion rests on an important assumption that is not universally valid. The security and correctness of function of the underlying operating systems must be suitably high so that the network security methods described here are not being built on an unreliable base, obviating their safety. Fortunately, reasonably secure operating systems are well on their way; so this intrinsic dependence of network security on appropriate operating system support should not seriously delay common carrier security [McCa79, Pope78, Feie79].

One could, however, take a rather different view of the nature of the network security problem. The goal might be to provide a high-level extended machine for the user, in which no explicit awareness of the network is required. The underlying facility is trusted to move data securely from site to site as necessary to support whatever data types and operations are relevant to the user. The facility operates securely and with integrity in the face of unplanned crashes of any nodes in the network. Synchronization of operations on user meaningful objects (such as operation *withdrawal* from object *checking account*) is reliably maintained using minimum trusted mechanism. Other higher level security-relevant operations beyond digital signatures are provided. If one takes such a high-level view of the goal of network security, then the simple common-carrier solutions are insufficient and more work remains.

ACKNOWLEDGMENTS

The authors thank the referees for their comments. In particular, it is a pleasure to acknowledge Adele Goldberg for her help and guidance in revising the manuscript.

BIBLIOGRAPHY

Aho74 Aho, A., Hopcroft, J., and Ullman, J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

Bogg80 Boggs, D., Shoch, J., Taft, E., and Metcalfe, R., "Pup: An internetwork architecture," to appear in *IEEE Trans. Comput.*, Feb. 1980.

Bran73 Branstad, D. K., "Security aspects of computer networks," presented at the AIAA Computer Network Systems Conf, April 1973.

Bran75 Branstad, D. K., "Encryption protection in computer data communications," in *Proc. 4th Data Communications Symp.*, 1975, pp. 8-1–8-7.

Carl75 Carlstedt, J., Bisbey, R., and Popek, G, *Pattern directed protection evaluation*, Rep. ISI/RR-75-31, Information Sciences Inst., U. of Southern California, Marina Del Rey, Calif., 1975.

Cerf78 Cerf, V., and Kirstein, P., "Issues in packet-network interconnection," *Proc. IEEE*, **66**, 11 (Nov. 1978), 1386–1408.

Denn66 Dennis, J B., and Van Horn, E. C., "Programming semantics for multiprogrammed computations," *Commun. ACM* **9**, 3 (March 1966), 143–155.

Diff76a Diffie, W., and Hellman, M, "Multiuser cryptographic techniques," in *Proc. 1976 AFIPS NCC*, Vol. 45, AFIPS Press, Arlington, Va., pp. 109–112.

Diff76b Diffie, W., and Hellman, M, "New directions in cryptography," *IEEE Trans. Inf. Theory* **IT-22**, 11 (Nov. 1976), 644–654.

Diff77 Diffie, W., and Hellman, M., "Exhaustive cryptanalysis of the NBS data encryption standard," *Computer* **10**, 6 (June 1977), 74–84.

Diff79 Diffie, W., and Hellman, M., "Privacy and authentication: An introduction to cryptography," *Proc. IEEE* **67**, 3 (March 1979), 397–427.

Down79 Downs, D., and Popek, G., "Data base system security and Ingres," in *Proc. Conf. Very Large Data Bases*, 1979, Rio De Janeiro, Brazil.

Evan74 Evans, A., Kantrowitz, W., and Weiss, E., "A user authentication system not requiring secrecy in the computer," *Commun. ACM* **17**, 8 (Aug. 1974), 437–442

Fabr74 Fabry, R. S, "Capability-based addressing," *Commun. ACM* **17**, 7 (July 1974), 403–412.

Feie79 Feiertag, R, and Neumann, P., "The foundations of a provably secure operating system (PSOS)," in *Proc. 1979 AFIPS NCC*, Vol. 48, AFIPS Press, Arlington, Va, pp. 329–334.

FEIS73 FEISTEL, H., "Cryptography and computer privacy," *Sc. Am.* **228,** 5 (May 1973), 15-23.

FEIS75 FEISTEL, H., NOTZ, W A., AND SMITH, J. L., "Some cryptographic techniques for machine to machine data communications," *Proc. IEEE* **63,** 11 (Nov. 1975), 1545-1554.

GAIN77 GAINES, R. S., Private communication, Sept. 1977.

GOOD77 GOOD, D. I., "Constructing verified and reliable communications processing systems," *ACM Soft. Eng. Notes*, Oct 77, pp. 2-5.

HELL78 HELLMAN, M. E., "Security in communications networks," in *Proc. 1978 AFIPS NCC*, Vol. 47, AFIPS Press, Arlington, Va., pp 1131-1134.

KENT76 KENT, S., *Encryption-based protection protocols for interactive user-computer communication*, Tech. Rep. 162, M I T. Lab. for Computer Science, May 1976.

KIMB75 KIMBLETON, S. R , AND SCHNEIDER, G. M , "Computer communications networks Approaches, objectives and performance considerations," *Comput. Surv.* **7,** 3 (Sept 1975), 129-173.

KLIN79 KLINE, C S., AND POPEK, G. J., "Public key vs. conventional key encryption," in *Proc. 1979 AFIPS NCC*, Vol. 48, AFIPS Press, Arlington, Va., pp. 831-837

KOHN78 KOHNFELDER, L., *Towards a practical public-key cryptosystem*, B.S. Thesis, M.I.T , Cambridge, Mass.

LAMP73 LAMPSON, B. W., "A note on the confinement problem," *Commun. ACM* **16,** 10 (Oct 1973), 613- 615

LEMP79 LEMPEL, A , "Cryptology in transition," *Comput. Surv.* **11,** 4 (Dec. 1979), 285-304.

LENN78 LENNON, R. E , "Cryptography architecture for information security," *IBM Syst. J* **17,** 2 (1978), 138-150.

MATY78 MATYAS, S M., AND MEYER, C. H., "Generation, distribution, and installation of cryptographic keys," *IBM Syst. J* **17,** 2 (1978), 126-137.

MCCA79 MCCAULEY, E. J , AND DRONGOWSKI, P J., "KSOS—The design of a secure operating system," in *Proc 1979 AFIPS NCC*, Vol 48, AFIPS Press, Arlington, Va., pp. 345-353.

MERK78 MERKLE, R. C , "Secure communications over insecure channels," *Commun. ACM* **21,** 4 (April 1978), 294-299.

MEYE73 MEYER, C. H , "Design considerations for cryptography, in *Proc. 1973 AFIPS NCC*, Vol. 42, AFIPS Press, Arlington, Va., pp. 603-606.

NBS77 NATIONAL BUREAU OF STANDARDS, *Data encryption standard*, Federal Information Processing Standards Publ 46, 1977

NBS78a NATIONAL BUREAU OF STANDARDS, *Design alternatives for computer network security*, Special Publ. 500-21, Vol. 1, NBS, Washington, D.C., 1978.

NBS78b NATIONAL BUREAU OF STANDARDS, *The network security center: A system level approach to computer security*, Special Publ. 500-21, Vol 1, Washington, D.C., 1978.

NEED78 NEEDHAM, R M., AND SCHROEDER, M D , "Using encryption for authentication in large networks of computers," *Commun. ACM* **21,** 12 (Dec 1978), 993-999.

POPE74a POPEK, G J., "Protection structures," *IEEE Comput.* (July 1974), 22-33

POPE74b POPEK, G J., "A principle of kernel design," in *Proc. 1974 AFIPS NCC*, Vol. 43, AFIPS Press, Arlington, Va., pp. 977-978.

POPE78 POPEK, G. J., AND KLINE, C S., "Design issues for secure computer networks," in *Operating Systems, an Advanced Course*, R. Bayer, R. M. Graham, and G. Seegmuller, Eds , Springer-Verlag, New York, 1978.

POPE79 POPEK, G. J., KAMPE, M , KLINE, C. S., STOUGHTON, A., URBAN, M , AND WALTON, E. J., "UCLA secure unix," in *Proc. 1979 AFIPS NCC*, Vol 48, AFIPS Press, Arlington, Va., pp 355-364.

RABI78 RABIN, M., "Digitalized signatures," in *Foundations of Secure Computing*, R Demillo et al., Eds., Academic Press, New York, 1978.

REDE74 REDELL, D D , AND FABRY, R S., "Selective revocation of capabilities," in *Proc IRIA Conf. Protection in Operating Systems*, Rocquencourt, France, Aug. 1974

RIVE77a RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L., *A method for obtaining digital signatures and public-key cryptosystems*, Tech. Memo. LCS/TM82, M I.T Lab. for Computer Science, Cambridge, Mass., April 4, 1977 (revised Aug 31, 1977).

RIVE78 RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L., "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM* **21,** 2 (Feb 1978), 120-126.

ROBE73 ROBERTS, L., AND WESSLER, B., "The ARPA network," in *Computer Communication Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1973, pp. 485-499.

SALT78 SALTZER, J , "On digital signatures," *ACM Operating Syst. Rev.* **12,** 2 (Apr. 1978), 12-14.

SEND78 SENDROW, M., "Key management in EFT environments," in *Proc. COMPCON 1978*, pp. 351-354 (available from IEEE, New York).

SIMM79 SIMMONS, G. J., "The asymmetric encryption/decryption channel," *Comput. Surv* **11,** 4 (Dec. 1979), 305-330

WEST70 WESTIN, A. F., *Privacy and Freedom*, Atheneum Press, New York, 1970.

RECEIVED APRIL 1979; FINAL REVISION ACCEPTED SEPTEMBER 1979