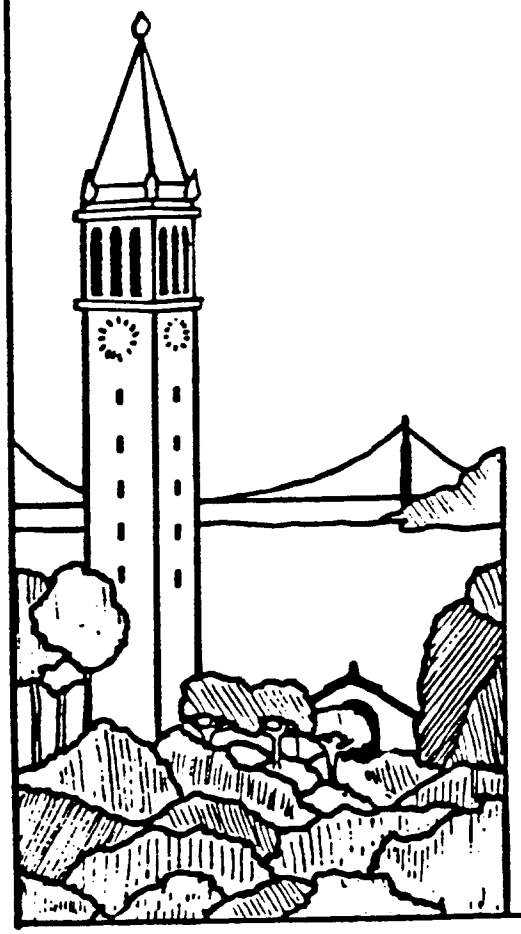
328

A Basis for Secure Communication in Large Distributed Systems

David P. Anderson
P. Venkat Rangan



Report No. UCB/CSD 87/328

January 1987
PROGRES Report No. 86.5

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A Basis for Secure Communication in Large Distributed Systems

David P. Anderson
P. Venkat Rangan

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

Abstract

Large distributed systems have properties that, from the point of view of security, distinguish them from LAN-based systems. We describe these differences, and show that the security mechanisms found in current distributed systems are not well-suited to large systems. We propose a secure communication architecture for large systems that puts security below the transport level. We argue that this is preferable to putting it at higher levels, and that in fact it can simplify and improve the performance of transport protocols.

1. Introduction

Future wide-area data networks will offer end-to-end bandwidth exceeding that available on current local-area networks [24]. We hypothesize that these networks will be used to build *large-scale distributed systems* having the following properties:

- Sharing of resources (processing power, data, communication services) over long distances is possible, is well-integrated (perhaps transparently) in the system, and is efficient.
- The system spans individuals and organizations that may wish to share resources, but that do not trust each other. They demand strict control over their own resources and ability to function autonomously.
- Resource access uses diverse communication protocols, including both request/reply and stream-oriented protocols.

Sponsored by MICRO, IBM, Olivetti, MICOM-Interlan, Defense Advanced Research Projects Agency (DoD) Arpa Order No. 4871 Monitored by Naval Electronic Systems Command under Contract No. N00039-84-C-0089. Venkat Rangan is also supported by an IBM Fellowship

- **Hardware components in general have no physical security. Hosts, including those acting as network gateways, can be loaded with arbitrary kernel software by malicious users.**

Most existing and proposed distributed systems make security-related assumptions that are incompatible with one or more of the above properties of large-scale systems. A sampling of these assumptions:

- All system-level components trust one another. Once a user is authenticated to the local kernel (via a password mechanism, perhaps) access to remote resources requires no further authentication. Examples are V [11], Eden [3] and Cronus [4].
- The system is controlled by a single administrative agent, so that it is possible to have universally trusted name and/or key servers. Examples are Sun UNIX [25], Mach [23], and Grapevine [9,10].
- The system is restricted to a single local area network that, like the Ethernet [19], guarantees that either every host on the network correctly receives a packet, or that no host receives it. Thus a packet cannot be modified by a malicious host. The system may span networks connected by gateways if these gateways are physically secure.
- There is a single administrative agent that can punish security violators, e.g. by firing them. Therefore detection, rather than prevention, is sufficient.
- Resource sharing across administrative domains is possible, but is poorly integrated (e.g. is restricted to mail and file transfer). Typically there are no global naming or authentication mechanisms, and a user must have independent accounts and passwords on each host. An example is Berkeley UNIX [6].
- The network is physically secure, or the network interfaces are assumed to contain tamper-proof logic, as in Amoeba [20].

To reiterate, a large-scale distributed system, as we have defined it, fails to satisfy any of the above assumptions, and thus has security problems that are not present in current distributed systems.

In this paper, we propose a mechanism for secure communication in large-scale distributed systems. The mechanism is called *Authenticated Datagram Protocol* (ADP). It is positioned below the transport layer, at the level of datagram communication between hosts. Potentially all network communication passes through ADP. We are currently building a distributed system called DASH that uses ADP [7].

ADP will be shown to have these properties:

- It satisfies the requirements of large-scale distributed systems.
- It can provide several levels of end-to-end security.
- It has several advantages over designs in which the security mechanism is at a higher level of the protocol hierarchy.

The rest of the paper is organized as follows. Section 2 describes the abstraction offered by ADP, section 3 gives its design, and section 4 discusses its implementation. Section 5 compares ADP to other designs for secure network communication, and section 6 offers some conclusions.

2. The ADP Abstraction and Interface

2.1. The Security Model

Security principals in ADP are called *owners*. Each owner has a unique symbolic name and a unique private/public-key pair. The mechanisms by which naming authority is delegated and by which name to public-key mapping is done are outside the scope of this paper. We assume, however, that they are done in such a way that an individual or organization can control both its own naming and the resolution of outside names.

The system consists of a set of *hosts*, each running an operating system *kernel*. At any point, a host has a *kernel owner*. The ownership of a kernel is established at boot time, before network communication takes place; it might be done manually or from a ROM.

The kernel may support multiple *user processes*, each of which has an associated owner, perhaps different from the kernel owner. A kernel possesses the private keys of its kernel owner and of all owners of user processes it has executed.

Kernel ownership may change over time, e.g. as different people boot a public workstation. A crash-free period under a single kernel owner is called a *kernel session*.

A kernel is *security-correct* if

(1) It limits information flow out of local processes to that explicitly requested by the processes.

(2) It performs name resolution correctly.

(3) It correctly executes the algorithms of ADP, as described below.

(4) Only the owner of the kernel can read the secret keys it contains.

These conditions all require some form of logical correctness on the part of the kernel; methods of ensuring this are outside the scope of this paper (see [5]). Condition 4 is met if either of the following holds:

a) The host is physically accessible only to its current owner.

b) There is no mechanism for reading the kernel's secret key storage. This is not the case if there is a "reset button" that jumps into a ROM monitor without first destroying the keys, or if a hardware failure generates a memory dump to a file or device readable by anyone other than the kernel owner.

An owner X is *kernel-trustworthy* if whenever X owns a kernel, it is security-correct.

An owner X is placing *kernel trust* in an owner Y if X executes user processes on a host with kernel owner Y.

2.2. The Function of ADP

ADP is a kernel module that provides an authenticated datagram service to the rest of the kernel. It does not implement any particular authorization scheme nor does it provide other security guarantees such as confinement or freedom from denial of service. With ADP as a basis, the kernel may provide higher-level services for user processes. For example, DASH supports a general client/server model in which ADP underlies both user access protocols and kernel-level protocols for service location and authorization.

ADP allows suppression of message replays beyond a known time, and automatically eliminates duplicate delivery of messages, in the case where the underlying network generates duplicates. It does not provide reliable or sequenced message delivery.

ADP demultiplexes messages on the basis of *ports*, which are kernel-level communication endpoints. Each port has an ID that is unique over a kernel session.

ADP provides the following interfaces to the kernel:

2.2.1. Message Reception

A client can indicate to ADP that it is willing to receive messages on a port using

```
register_port(
        port: port_ID;          /* port being registered */
        local_owner: name;      /* owner of port */
        max_age: time;          /* bound for replay elimination */
)
```

ADP can then deliver messages to the port. It will tag each such message with the name of its sender. On each host, there is a distinguished port that is owned by the kernel owner and has a well-known port ID. In DASH, this port is used for kernel-level communication that initiates user-level communication.

ADP provides authentication; more precisely, when ADP delivers a message tagged with the owner-name X, it guarantees that the following hold:

- If *X* has placed kernel-trust only in kernel-trustworthy owners, then the message resulted from a call to **ADP_send** (see below) at the request of a process owned by *X*, and was directed to this port. Furthermore, if the *max_age* argument was present on port registration, the message was generated no more than that long ago.

2.2.2. Message Sending

The ADP primitive to send a secure message is:

```
ADP_send(
        message: string;
        local_owner: name;
        remote_host: name;
        remote_port: port_ID;
        privacy: boolean;
        max_delay: time;
)
```

ADP_send sends the given message to the named port on the destination host. *Local_owner* must be currently running a process on this host.

If the owner of the remote host is kernel-trustworthy, and the named port exists, then the message will be delivered only to that port, and will be tagged with the name of the sending owner.

ADP optionally provides privacy: more precisely, ADP guarantees that if the calling kernel is security-correct, the following holds:

If the *privacy* flag is set, the text of the message will be readable only by kernels with the same owner as the remote host.

The *max_delay* argument is a hint to the kernel that the message can be buffered for up to that amount of time before it is transmitted. This is used to encourage encryption piggybacking (see section 3.1).

ADP also provides the following primitives:

flag = **establish_channel**(remote_kernel_owner, remote_host: name;)

flag = **local_authenticate**(remote_owner, remote_host: name;)

flag = **remote_authenticate**(local_owner, remote_host, remote_owner: name;)

These are used by the kernel to do boot-time "advance authentication". **Establish_channel** returns true iff the remote owner claims ownership of the named host, and accepts a secure channel (see section 3.2.1). **Local_authenticate** and **remote_authenticate** authenticate a local owner to a remote host, and vice versa.

2.3. End-to-end Security

ADP satisfies the requirements of large-scale systems. In particular, it does not require kernels to trust each other or owners to trust all kernels or physical security. Neither does it restrict sharing of resources in any way. Other functions like authorization can be efficiently built on top of ADP [7]. Properties of ADP related to other requirements of large-scale systems are discussed in section 4.

In our model of distributed computation, an owner who runs a process on a kernel with a different owner has no privacy from that owner. If the kernel is not security-correct, it can alter or publicize any data accessible to the user process. Hence no additional security is obtained by doing encryption at higher kernel levels or in user processes. The host-to-host security provided by ADP, together with the security-correctness of the kernel, are, in our model, as much end-to-end security as is possible without using external mechanisms. As will be shown in section 3.3, ADP can also be used to provide other levels of end-to-end security.

3. The Design of ADP

The design of ADP can be summarized as follows: it uses host-to-host secure channels and demand-driven authentication, caching authentication information where possible (see Figure 1).

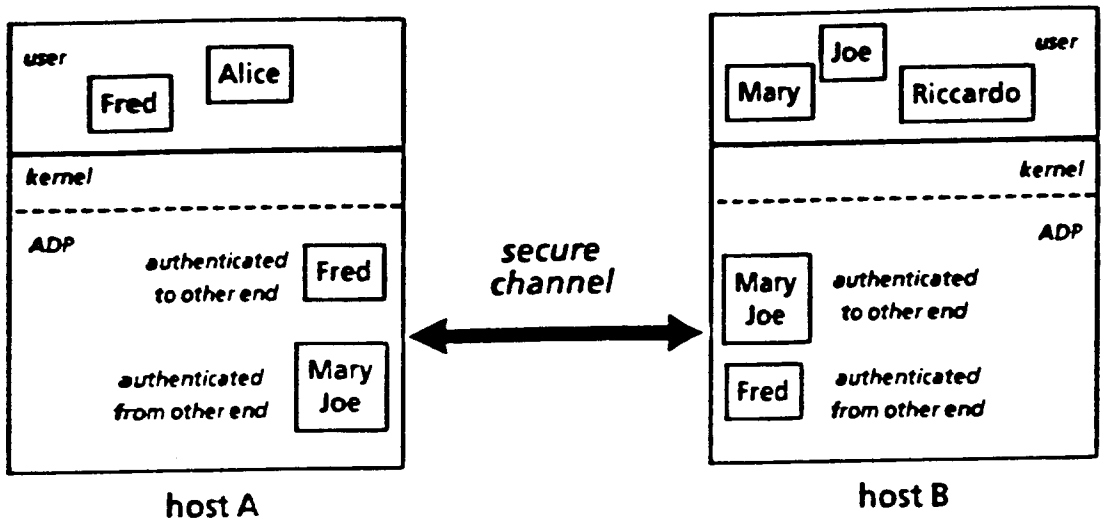


Figure 1. Authentication Caching in ADP.

The ADP module on each host maintains, for each secure channel, two lists of owners:

(1) Owners that it has authenticated to the other end of the channel.

(2) Owners authenticated to it by the other end of the channel.

This *authentication caching* means that expensive PKE-based authentication need be done only the first time an owner is involved as a sender or receiver on a particular secure channel.

In many cases, patterns of communication (in terms of local and remote owners and addresses) are predictable; for example, a workstation will always communicate with local file servers. It is then possible for a kernel to establish secure channels, and do authentication on those channels, in advance of user demand (e.g. at boot time).

The security functions of ADP reduce to secure channels and owner authentication. In general, both must be implemented cryptographically; in some cases, a cheaper solution may be possible.

3.1. Messages

ADP makes use of an underlying network layer that provides an insecure datagram service. This could vary according to the remote host involved; for example, the Internet Protocol might be used [26] for a distant host, while a simpler network protocol would suffice for a host on the same LAN. In some cases ADP must handle fragmentation of long messages; this will not be considered here.

An ADP message consists of an *ADP header* and zero or more *user messages*. The ADP header can contain a secure channel request or acknowledgement, zero or more "authentication signatures", zero or more authentication requests, a sequence number, a timestamp (used for replay elimination), and a cryptographic checksum [2,15] of the entire ADP message. Each user message has a header containing the port number, and the message data. A detailed description of the ADP message format can be found in [8].

If user messages do not require privacy, encryption can be limited to the checksum in the ADP header. Furthermore, on a broadcast LAN like Ethernet the checksum can be replaced by an encrypted sequence number. In either case, it may be possible to *piggyback* multiple user messages into a single ADP message, thereby reducing the encryption cost per user message. The *max_delay* parameter of **ADP_send** allows ADP clients to indicate that messages can be delayed, thereby encouraging piggybacking.

ADP does duplicate elimination by putting sequence numbers in ADP messages, and maintaining a bitmask per secure channel describing the recent history of received messages. Any messages before the start of the bitmask are ignored. Timestamps are used to eliminate "replays", i.e. messages removed by a malicious kernel and resent after a long delay.

3.2. Cryptographic Implementation

Cryptography can be used to provide secure channels and for authentication (using signatures). Our implementation uses a bootstrapping mechanism to combine the advantages of public-key [16,22] and single-key [21] cryptography.

As a basis for authentication in large distributed systems, public-key schemes have several advantages compared to single-key schemes [17]. In large-scale distributed systems replication is essential for performance, availability, and fault-tolerance. Key server replication in a single-key scheme increases its vulnerability to attack on secrecy, whereas it reduces the vulnerability of public-key systems.

Current public-key encryption algorithms are too slow to consider using them to encrypt each message sent. On the other hand, single-key operations are fast enough to be employed for each message. In ADP a public-key scheme is used to bootstrap into a single-key scheme.

3.2.1. Cryptographic Implementation of Secure Channels

When a host X needs to establish a secure channel to a host Y, it sends a channel establishment request to Y. This request contains random strings S and T encrypted with the public key of the owner of Y. S will be used as the secret single-key of the secure channel, and T will be used to authenticate owners from Y to X. X marks the secure channel as being *tentative* until it receives an acknowledgement. The secure channel request is included with ADP messages sent while the channel is still tentative.

Y sends to X a random string R to be used for signatures sent from X to Y in the secure channel acknowledgement and in every ADP message until the first signature is received.

If two hosts simultaneously try to establish a channel with each other, the one with the lexicographically greater name determines the channel key.

3.2.2. Cryptographic Implementation of Owner Authentication

Public-key encryption is used for signatures [1,13,14]. An owner's signature is the random string agreed upon during secure channel establishment, encrypted with the private key of the owner. To check the validity of a signature, the receiver of a signature obtains the public key of the owner from the name server, decrypts the signature with the public key, and checks if the resulting string is the correct random string.

3.3. Trust Domains

Suppose a group of hosts and the communication channels between them are physically secure, and that the owners with access to the hosts all place kernel-trust in one another.

Call such a group a *trust domain*. Encryption-based security mechanisms are necessary only for communication across the domain boundary. Suppose also that all communication across the boundary is routed through one or more hosts called *domain gateways*. Then it is possible to have a special ADP module on a domain gateway that handles packet forwarding. It handles secure channels and authentication on behalf of hosts within the domain, transparently to kernel and user level clients and to higher level protocols. This has the following advantages:

- Efficiency: communication within the domain has no security overhead. Only the domain gateway does encryption, so only it need have encryption hardware.
- Flexibility: domain configuration may be changed at any time. Only the implementation of secure channels and owner authentication changes. Kernel and user clients, and higher level protocols do not see any changes.

4. Comparison with Other Architectures for Secure Communication

We have proposed putting security in a layer just above (or part of) the network layer. We now list the advantages of this approach in three parts:

1) general advantages of ADP
2) specific advantages relative to transport layer security
3) specific advantages relative to upper layer security

4.1. General Advantages of ADP

Putting security at the level of host to host datagrams has several advantages:

- It simplifies transport level protocols. When a host crashes, its secure channels are destroyed. Thus remote host crashes are detected at the host-to-host level at the time of secure channel establishment. In combination with the elimination of duplicates and the limiting of replay delay, this means that 3-way handshakes can often be eliminated from transport-level protocols.
- Three-way handshakes can be eliminated from stream and request/reply protocols. A short transaction then requires just two messages, as opposed to at least 6 in TCP and 4 in secure RPC.
- Security functions need not be duplicated in multiple transport protocols or user programs.
- There are two public-key operations per owner per remote host per kernel session. Often these operations can be done at boot time or during idle periods. There are no per-process or per-operation public-key operations, resulting in a substantial performance gain.
- As was shown in section 3, security at the host-to-host datagram level allows heterogeneity in implementation and flexibility to change the implementation without the need to change any of the higher level protocols.
- Since messages from all client processes and higher level protocols pass through ADP, a number of these messages destined to a common remote host can all be combined into a single datagram and authenticated once using the channel secret key. This can reduce the number of single key operations.

4.2. Advantages over Transport Layer Security

Secure RPC [10], SUN secure RPC [25], and secure TCP [18] involve the addition of a security mechanism to an existing communication protocol. Two popular communication paradigms are request/reply [12] and full duplex byte stream [27]. We examine secure RPC as an instance of security in the request/reply paradigm and secure TCP as an instance of security in the full duplex byte stream paradigm.

4.2.1. Secure RPC

When a client issues its first RPC request to a remote server, the RPC mechanism establishes a "secure conversation" between the two processes. This consists of agreeing on a secret conversation key to be used for encrypting RPC requests and replies. There are several disadvantages of such a scheme:

- For each conversation, the RPC system must maintain a long-term state consisting of a conversation key and the sequence number of requests within a conversation.
- A 3-way handshake is necessary to agree upon the conversation key. The cost of this 3-way handshake is small if it is amortized over many RPC's. If, however, there are lots of short-lived processes making just one or two remote procedure calls the performance penalty due to a 3-way handshake is substantial. This can reduce the efficiency of RPC for short transactions.
- If public keys are used to authenticate the client and the server processes to each other, there will be four public-key operations for each conversation. If the conversation consists of a single RPC, the relative cost is substantial.
- There is a single-key encryption and a decryption for each RPC request, reply, and acknowledgement. Since messages from different processes use different secure channel keys, it is not possible to reduce encryption cost by piggybacking messages from different processes that are all destined to the same host.

4.2.2. Secure TCP

TCP is a DARPA Internet transport protocol [27] providing full duplex byte stream connections between processes on different hosts. The secure TCP design described in [18] consists of an initial agreement upon a secret key to be used during the TCP connection after the end points are authenticated to each other. There are several disadvantages associated with this scheme:

- A TCP connection is between ports whereas the principals in authentication are users. Existing TCP architecture would require modification to incorporate the notion of users.
- Four public-key operations are performed for each TCP connection.
- Encryption cost reduction by piggybacking is impossible since keys are not per host-pair.

4.3. Advantages over Upper-level Security

There are disadvantages in adding secure communication mechanisms above the transport level:

- Transport level protocols like TCP do connection establishment using 3-way handshakes. A secure communication mechanism requires its own handshake to agree upon keys after the transport level has established its connection. This duplication of handshaking entails a high message overhead.
- Transport level protocols do error detection using (insecure) checksums. Secure communication mechanisms above do their own cryptographic checksumming. This is an unnecessary duplication of effort.
- Transport levels do sequencing. An intruder can change the transport level headers and hence the transport level sequence number. Thus an upper-level security mechanism must also do sequencing, again producing an unnecessary duplication of

effort.

- If an intruder sends a false message with the correct transport level sequence number, the transport level protocol will accept it as the next message and reject the true message which may arrive later. The secure communication mechanisms above will reject the false message correctly but will never get the true message. False acknowledgements at lower levels can disrupt the sequencing. The only way to recover from such situation is to re-establish the connection at both the transport and the secure communication levels. This has the potential for a lot of unnecessary tearing down of connections and the associated performance overhead.
- Replayed and unauthenticated messages are detected only at the upper level. These messages are unnecessarily processed at all lower levels of the protocol hierarchy.
- Public-key operations cannot be reduced because authentication is not per-host, and single-key operations cannot be reduced by piggybacking.

5. Conclusion

We have described the security requirements of large-scale distributed systems and have shown that existing mechanisms for secure communication do not address these needs. The ADP architecture offers efficient end-to-end secure communication in large-scale systems by providing authentication and privacy at the level of host-to-host datagrams. ADP has numerous advantages over designs in which security mechanisms are placed at other protocol levels.

ADP has been implemented as part of an ongoing research project in large-scale distributed systems (DASH). Experiments are currently being conducted to explore the performance of ADP in a wide range of message queueing policies and load conditions.

We would like to thank Domenico Ferrari, Shin-Yuan Tzou, Brian Bershad, Bruno Sartirana and Kevin Fall for their contributions to DASH and ADP.

6. References

[1] Selim G. Akl.

Digital Signatures: A Tutorial Survey.

IEEE Computer, pages 15-24, February 1983.

[2] Selim G. Akl.

On The Security of Compressed Encodings.

Advances in Cryptology: Proceedings of Crypto'83 , pages 209-230.

[3] G. T. Almes et al.

The Architecture of the Eden System.

IEEE Trans. on S/W Engg., SE-11(1), January 1985.

[4] R. Schantz et al.

Cronus, A Distributed Operating System.

BBN Laboratories Report No. 5885.

[5] Cheheyl et al.

Verifiying Security.

Computing Surveys, 13(3):276-337, September 1981.

[6] W. Joy et. al.

4.2BSD System Manual, Tech. Report, CSRG, CS Div., U.C. Berkeley, July 1983.

[7] D.P. Anderson, D. Ferrari, P. Venkat Rangan and S. Tzou.

Design of the DASH Distributed System.

University of California, Berkeley/Computer Science Department Technical Report, January 1987 .

[8] D.P. Anderson and P. Venkat Rangan.

ADP: An Authenticated Datagram Protocol for Large-Scale Distributed Systems.

University of California, Berkeley/Computer Science Department Technical Report, January 1987.

[9] A. D. Birrell, R. Levin, R. Needham and M. D. Schroeder.

Grapevine: An exercise in distributed computing.

Communications of the ACM, 25(4):260-274, April 1982.

[10] A. D. Birrell.

Secure Communication Using Remote Procedure Calls.

ACM Transactions on Computer Systems, 3(1):1-14, February 1985.

[11] D. R. Cheriton.

The V Kernel: A Software Base for Distributed Systems.

IEEE Software, pages 19-42, April 1984.

[12] D. R. Cheriton.

VMTP : A Transport Protocol for the Next Generation of Communication Systems.

Proceedings of the Data Communications Symposium, pages 406-415, Aug 1986.

[13] D. E. Denning.

Protecting public keys and signature keys.

IEEE Computer, 16(2):27-35, February 1983.

[14] Dorothy E. Denning.

Digital Signatures with RSA and Other Public-Key Cryptosystems.

Communications of the ACM, 27(4):388-392, April 1984.

[15] D. E. Denning.

Cryptographic Checksums for Multilevel Database Security.

Proceedings of the Symposium on Secuirty and Privacy, pages 52-61, May 1984.

[16] W. Diffie and M. Hellman.

New Directions in Cryptography.

IEEE Trans. Information Theory, IT-22(6):644-654, Nov 1976.

[17] W. Diffie.

Conventional Versus Public Key Cryptosystems.

Secure Communications and Asymetric Cryptosystems, Edited by Gustavus J. Simmons, Westview Press, Boulder, Colorado, 1982.

[18] W. Diffie.

Security for the DoD TCP.

CRYPTO , 1985.

[19] R. M. Metcalfe and D. R. Boggs.

Communications of the ACM, 19(7):395-403, July 1976.

[20] S. J. Mullender and A. S. Tanenbaum.

Protection and Resource Control in Distributed Operating Systems.

Computer Networks, 8:421-432, 1984.

[21] NBS.

Data Encryption Standard.

FIPS publication 46, NBS, U.S. Dept. of Commerce, Washington, D.C., 1977.

[22] R. L. Rivest. A. Shamir and L. Adleman.

A method for obtaining digital signatures and public-key cryptosystems.

Communications of the ACM, 21(2):120-126, February 1978.

[23] R. D. Sansom, D. P. Julin and R. F. Rashid.

Extending a Capability Based System into a Network Environment.

Technical Report, Computer Science Department, Carnegie-Mellon University, April 1986.

[24] E.L. Slate and J.A. Popko.

The Next Five Years in Communications.

Telecommunications, pages 49-60, January 1986.

[25] B. Taylor and D. Goldberg.

Secure Networking in the Sun Environment.

Summer Usenix, pages 27-36, 1986.

[26]

DoD Standard, Internet Protocol.

Information Sciences Institute, University of Southern California, Marina del Rey, California. RFC 791, September 1981.

[27]

DoD Standard, Transmission Control Protocol.

Information Sciences Institute, University of Southern California, Marina del Rey, California. RFC 793, September 1981.