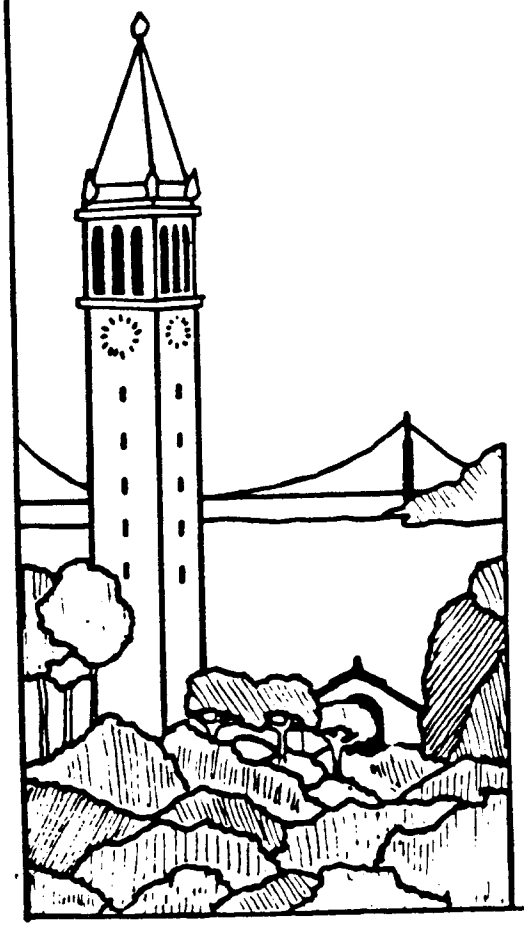
Subtransport Level: The Right Place for End-to-End Security Mechanisms

David P. Anderson

Domenico Ferrari

P. Venkat Rangan



Report No. UCB/CSD 87/346

March 1987
PROGRES Report No. 87.2

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

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 1987		**00-00-1987 to 00-00-1987**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Subtransport Level: The Right Place for End-to-End Security Mechanisms	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Department of Electrical Engineering and Computer Sciences,Berkeley,CA,94720**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We argue that end-to-end authentication and privacy in loosely-coupled distributed systems are not only achievable by mechanisms at the host-to-host (i.e., subtransport) level under generally satisfiable conditions, but that this solution can be more advantageous than those based on security mechanisms at higher levels of the protocol hierarchy in terms of both functionality and performance. We introduce a model of communication security and a subtransport-level protocol called ADP (the Authenticated Datagram Protocol), which provides end-to-end authentication and privacy consistently with the definitions of the model. We then discuss the advantages of the subtransport approach, and present some experimental results from the measurement of a prototype of ADP that confirm the expected performance benefits of this approach.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**20**	

Subtransport Level: The Right Place for End-to-End Security Mechanisms

David P. Anderson, Domenico Ferrari, and P. Venkat Rangan

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

We argue that end-to-end authentication and privacy in loosely-coupled distributed systems are not only achievable by mechanisms at the host-to-host (i.e., subtransport) level under generally satisfiable conditions, but that this solution can be more advantageous than those based on security mechanisms at higher levels of the protocol hierarchy in terms of both functionality and performance. We introduce a model of communication security and a subtransport-level protocol called ADP (the Authenticated Datagram Protocol), which provides end-to-end authentication and privacy consistently with the definitions of the model. We then discuss the advantages of the subtransport approach, and present some experimental results from the measurement of a prototype of ADP that confirm the expected performance benefits of this approach.

This research was supported by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 4871, monitored by the Naval Electronic Systems Command under Contract No. N00039-84-C-0089, by the IBM Corporation, by Olivetti S.p.A., by MICOM-Interlan, Inc., by CSELT S.p.A., and by the University of California under the MICRO Program. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of any of the sponsoring agencies or corporations.

1. Introduction

Security mechanisms can be introduced at various levels in a network protocol architecture. In special cases, introducing them at more than one level may be justified. However, to satisfy the two communication security goals of authentication of principals and message privacy, it is possible to add security mechanisms to only one layer in the protocol hierarchy.

What layer should this be? Many authors argue that security should be at the transport level (e.g., [13] [5]) or in any event not below it (e.g., [16]). The three reasons mentioned by Saltzer *et al.* [16] for placing security mechanisms above the communication subsystem (i.e., above the bottom three layers in the ISO Open Systems Interconnection model) are that

(a) with security at a higher level, the communication subsystem need not be "trusted to securely manage the required encryption keys;"

(b) if the security mechanisms were in the communication subsystem, the data would be "in the clear and thus vulnerable as they pass into the target node and are farmed out to the target application;"

(c) "the *authenticity* of the message must still be checked by the application."

Argument (a) is also made, though in more general terms, by Diffie [13], who favors locating the mechanisms at the transport level, "the lowest point of full end-to-end communication," since this provides the benefits of end-to-end security with a single mechanism. Birrell [5] states that the "lowest layer at which we can provide an end-to-end guarantee is the network layer," but "in most communication architectures (including ours) it is not until the transport layer that end-to-end security is feasible. The transport layer is the lowest level at which enough state information is kept to establish the authenticity of incoming data in successive packets of an interaction." Voydock and Kent [18] state that one of the alternative ways in which end-to-end security can be achieved is the host-to-host solution, but add that end-to-end security measures "usually extend beyond the communication subnet." They also do not hide their predilection for association-oriented mechanisms, that "constitute a refinement of end-to-end measures," and are restricted to layers 4 through 7 of the ISO OSI model. Tanenbaum [17] states that logically the encryption should be at the presentation layer and that "for historical reasons and implementation convenience, however, it is often put elsewhere, typically the transport layer or the data link layer".

In this paper, we argue that end-to-end security is not only feasible at the host-to-host (or *subtransport*) level under generally satisfiable conditions, but that this solution can be more advantageous than the others in many respects, including that of performance, which is not usually referred to in the literature. To prove these claims, we introduce our model of communication security in Section 2, we describe a subtransport-level protocol called ADP (Authenticated Datagram Protocol) in Section 3, we show in Section 4 that ADP provides end-to-end security according to the model defined in Section 2, we discuss the advantages of the subtransport approach in Section 5, and in section 6 we present some experimental results that confirm the expected performance benefits of this approach.

2. The Security Model

The two goals of network security we are concerned with in this paper are message *authenticity* and message *privacy*. Both concepts can be defined by referring to the situation depicted in Figure 1. Each user process has an owner. Each kernel also has an owner. Owners are the security principals in our model. Each owner has a unique symbolic name

and a unique private-key/public-key pair. The owner of object O will be denoted in the sequel by (O). All communication protocols are assumed to be included in the box called "kernel" in Figure 1.

When process Y receives a message that is alleged to have been sent by the owner of process X running on A, (Y) wants to be sure of its authenticity, that is, that it was really sent by (X), even though (Y) may not trust (A) or the network (since Y runs on B, we assume that (Y) trusts (B)). Note that "the network" in the previous sentence and in Figure 1 stands for the path (or paths) possibly including hosts and gateways a message from X has to go through in order to reach Y.

When (X) wishes its message to (Y) to be private, it wants to ensure that the contents of the message can be read only by (Y), or (if the message is not delivered or delivered in a corrupted state) it cannot be read by anyone, even though (X) may not trust (B) or the network.

Security mechanisms to be incorporated into a protocol layer in each kernel should provide end-to-end authentication and privacy as we have just defined them. In order to prove (as we shall do in Section 4) that it is possible to achieve end-to-end security by incorporating suitable mechanisms below the transport layer, we shall introduce the notions of *security-correct kernel* and *kernel trust.*

Each kernel runs on a *host.* At any point in time, 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. 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.* In the sequel, we shall occasionally use the terms "host" and "kernel" interchangeably, as this can be done without confusion during each kernel session.

The kernel may support multiple *user processes,* each of which has an associated owner, perhaps different from the kernel owner. Processes communicate with each other through messages. Each message has a message sender field containing the name of the owner of the sending process, and a message receiver field, containing destination information. A kernel possesses the private keys of its kernel owner and of all owners of the user processes it has executed or is executing. Figure 2 represents the organization of the kernel. It consists of modules of code and private data. The passing of messages between modules is handled by a special kernel module called the *message passing module.* The network security functions are handled by a module called the *security module.* Kernel modules which handle either an outgoing message before it is passed to the security module or an incoming message after it has been processed by the security module are called *type-1 modules.* Protocol modules above the layer at which communication security functions are handled are examples of type-1 modules. Kernel modules which handle either an outgoing message after it has been processed by the security module or an incoming message after it has been processed by the security module are called *type-2 modules.* In a protocol architecture where security functions are handled above the data link layer, a network driver is an example of a type-2 module. Kernel modules other than the security module, the message passing module, the type-1 modules, and the type-2 modules are called *type-3 modules.* The private keys are part of the private data storage of the security module. The security module, the message passing module, the type-1 modules, and the type-2 modules are together called *critical modules.*

A kernel is *security-correct* if the following conditions hold:

(1) The only way for a user process to communicate with the kernel is through messages.

(2) When a user process sends a message to a kernel module, the message passing module sets the message sender field to be the owner of that process. Thereafter, the message passing module and the type-1 modules do not change (a) the message sender field, or (b) the message receiver field, (c) the data part of the message.

(3) The message passing module does not deliver a message to a user process if the owner of that user process is different from that indicated in the message receiver field.

(4) The private data storage of the security module is read or written by no other module.

(5) The security module executes its algorithms (to be given later) correctly.

(6) Type-2 modules do not directly communicate with type-1 or type-3 modules or user processes.

All of these conditions require some form of logical correctness on the part of the kernel; methods of ensuring this are outside the scope of this paper (see [6]).

An owner I is *kernel-trustworthy* if, whenever I owns a kernel, this kernel is security-correct.

If an owner I issues a command to execute a process (locally or remotely) on a kernel with owner J, we infer (and say) that I is placing *kernel trust* in J.

3. The Authenticated Datagram Protocol

As mentioned in Section 1, ADP is a host-to-host datagram protocol that provides secure communications, i.e., authentication and, optionally, privacy of messages. ADP separates authenticated principals from the end-points of a secure channel across a network.

ADP makes use of an underlying network layer that provides an insecure datagram service. This service could vary according to the remote host involved; for example, the Internet Protocol might be used [19] 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 issue will not be considered here.

ADP 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 (e.g., authorization mechanisms) for user processes.

3.1. Secure channel establishment

The operation of ADP is based on a host-to-host *secure channel* that is established whenever a (user or kernel) process executing on a host wants to communicate with another process running on another host and such a channel does not already exist. Public-key encryption (PKE) [11, 15] is used to establish a secure channel: when a host A needs to establish a secure channel to a host B, the ADP module running on A sends a channel establishment request to B. This request contains random string S, encrypted with the public key of the (B), and random string T. S will be used as the secret single key of the secure channel, and T will be used as a digital signature [1, 8, 10] to authenticate owners from B to A. A 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.

In the secure channel acknowledgement and in every ADP message until the first signature is received, B sends to A a random string R to be used for signatures sent from A to B, i.e., to authenticate owners from A to B. If the two hosts simultaneously try to establish a channel between themselves, the one with the lexicographically greater name determines the channel key S.

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).

3.2. Owner authentication

When an owner I who has not sent any messages to B before wants to communicate from host A with an owner J on host B, and a secure channel between A and B has been established, public-key encryption is used to authenticate I to (B). The ADP module running on A encrypts string R with the private key of I and sends it in a message along with I's name; the ADP module on B decrypts it with the public key of I obtained from the name server, and compares the result to string R; if the two strings are identical, then B concludes that the message is from I (or from a host that possesses I's private key), and caches I's name in the list of the owners authenticated from the other end that it maintains for that channel. A in turn caches the name of I in its list of owners authenticated to the other end of the channel (see Figure 3).

When this operation has been done, both hosts are aware, and remember, that I has been authenticated to (B). 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 all subsequent messages sent from I to J, the presence of I's name in the caches maintained by ADP on A and B is considered sufficient for the authentication of the messages.

3.3. Messages

As explained in [3], messages from I to J, when I has been authenticated to the host where the destination process belonging to Y resides, only require for the purposes of authentication either the encryption (with the secure channel's secret key S) of a message trailer that includes a sequence number or a cryptographic checksum [2, 9]. If privacy of the data is also desired, then the entire message may be encrypted with S. Thus, ADP uses a bootstrapping mechanism to combine the advantages of public-key [11, 15] and single-key [14] cryptography. As a basis for authentication in large distributed systems, public-key schemes have several advantages over single-key schemes. In those systems, replication is essential for performance, availability, and fault-tolerance. Key server replication increases the vulnerability of a single-key scheme to attack on secrecy, whereas it reduces the vulnerability of public-key systems [13]. Current public-key encryption algorithms are too slow to consider using them to encrypt any part of each message sent. On the other hand, single-key operations are fast enough to be employed for each message.

Further reductions in encryption overhead can be achieved if ADP and its users recognize the existence of messages that are not particularly urgent (e.g., most acknowledgements, most writes to remote disk) and that can therefore be delayed and perhaps piggybacked onto a later message to be transmitted over the same secure channel. In this case, one *ADP message* on the network may include a number of user messages. Only one sequence number will have to be encrypted, or only one cryptographic checksum will have to be computed, for each ADP message, thereby reducing the cost of encryption per user message.

The urgency of a message will typically be determined by the process originating it: a message marked urgent will be shipped by ADP as soon as possible; one whose maximum delay is specified will have to be sent when this timeout expires, unless it will have already been piggybacked onto an urgent message; finally, one which neither is marked urgent nor has its maximum delay specified will be assigned by ADP a timeout, which will usually be a tunable parameter of ADP and the same for all such messages. Since delayed messages have to be kept in queues within ADP, another tunable parameter is the maximum size of each queue: the arrival of a non-urgent message which causes a queue to fill or to overflow will be treated as if it were the arrival of an urgent message. This maximum size will depend on the amount of buffer space available, and may be limited by the maximum size

of the messages that can be accepted by the lower protocol layers. The one just described is the queue service discipline that has been implemented in the current version of ADP, but is by no means the only possible one, nor necessarily the best.

4. End-To-End Security in ADP

ADP provides an authenticated datagram service to transport-level protocols, and, optionally, message privacy over the network. To show that it does so according to the definitions given in Section 2, that is, that the authentication and privacy it provides are really end-to-end (i.e., owner-to-owner), it is useful to describe ADP's interfaces to the protocol layer above it.

4.1. ADP interfaces

4.1.1. Message reception

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. 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 for port /*
)
```

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. This port is used for request/reply style kernel-level communication that is used to set up other communication.

4.1.2. Message sending

The ADP primitive to send a 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 remote port on the destination 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 piggybacking (see Section 3.3).

4.1.3. Other ADP primitives

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;)

They can be 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. **Local_authenticate** and **remote_authenticate** authenticate a local owner to a remote host, and vice versa.

4.2. End-to-end security

We now prove that ADP provides end-to-end security between two owners I and J if every owner O in whom I or J has placed kernel trust is kernel-trustworthy.

If owner J has a process on a kernel B, since J has placed kernel trust in only kernel-trustworthy owners, kernel B is security-correct. Thus when the ADP module on kernel B delivers to a port owned by J a message tagged with the name of I, **ADP_receive** guarantees that the message arrived on a secure channel on which a signature of I had been received at an earlier time. From the assumption that I has placed kernel trust in only kernel-trustworthy owners, it can be concluded that the kernel A which sent the signature is security-correct. Thus both ends of the secure channel are security-correct and hence only the ADP modules on the two kernels know the secret key of the channel. Since the message was received on such a secure channel, it must have been sent by the ADP module on A. Since A is security-correct, the message must have resulted from some process owned by I on kernel A. Since receiving kernel B is also security-correct, the process owned by J with which the receiving port is associated is correctly informed about the identity of I. Thus, end-to-end authentication between owners, who are the security principals in our model, is guaranteed.

If a process owned by I on a kernel A desires privacy for a message to be sent to J, it first invokes **remote_authenticate** to make sure that J is actually on the remote host B. It then calls ADP_send with the privacy flag set. The authentication of J, together with the assumption that J has placed kernel trust only in kernel-trustworthy owners, guarantees that B is security-correct and that J has a process on B. Since I has placed kernel trust only in kernel-trustworthy owners, A is security-correct. The secure channel between the two security-correct kernels A and A guarantees privacy between the ADP modules on the two kernels. The security-correctness of A guarantees privacy from other owners having processes on A. The security-correctness of the receiving kernel guarantees privacy from other owners having processes on B. Thus we have end-to-end privacy between the two owners I and J.

If a kernel is not security-correct, it can alter or publicize any data accessible to a user process. In such a case, there can be no security guarantees, and no additional security is obtained by doing encryption at higher protocol layers or in user processes. If a kernel is security-correct, doing encryption within higher layers or in user processes may change some type-1 modules into type-2 modules as defined in our security model.

Obtaining process-to-process security from owner-to-owner security requires that, in addition to being security-correct, a kernel must limit the read/write access to a port (a message end-point) to the process which owns the port. Since this condition involves only processes on a single kernel, the special privileges of a kernel are sufficient to satisfy it.

4.3. Trust domains

The set of hosts in many distributed computing environments may contain subsets with the following property: within a subset, the hosts and the communication channels between them are physically secure, and owners with access to the hosts all place kernel-

trust in one another. Across subsets, the communication links may not be physically secure, and owners in one subset may not trust those in the other. We call such subsets *trust domains*. Encryption-based security mechanisms are necessary only for communication across a trust domain boundary. Suppose also that all communication across a boundary is routed through one or more hosts called *domain gateways*. Then it is possible to have a special subtransport protocol module on a domain gateway that handles packet forwarding. This module can also handle secure channels and authentication on behalf of hosts within the domain, transparently to kernel and user level clients and to higher level protocols (however, owners are still the principals being authenticated on a domain-to-domain secure channel). 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: the domain's configuration may be changed at any time. Only the implementation of secure channels and owner authentication in the domain gateways changes. Kernel and user clients, and higher level protocols do not see any changes.

5. Subtransport Versus Higher-Level Security

It was shown in the previous section that putting security at the subtransport layer provides the desired end-to-end authentication and privacy. We now discuss the advantages of putting security at the subtransport layer rather than at or above the transport layer. The advantages are grouped into the following three parts: 1) general advantages of subtransport layer security 2) specific advantages relative to transport layer security 3) specific advantages relative to putting security above the transport layer.

5.1. General advantages of the subtransport approach

Putting security at the subtransport layer has several advantages relative to putting it at higher protocol layers:

- It simplifies transport level protocols. When a host crashes, its secure channels are destroyed. Thus remote host crashes can be detected at the host-to-host level at the time of secure channel establishment, and transport level protocols do not have to employ elaborate timer mechanisms to detect them [7, 20]. This also means that 3-way handshakes can often be eliminated from transport-level protocols. A short transaction then requires just two messages in the best case, as opposed to at least six in TCP and four in secure RPC.
- At the subtransport level, security functions do not have to be duplicated as they have to be in higher layers.
- 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.
- Since messages from all client processes and higher level protocols pass through the subtransport layer, 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.
- As was shown in section 4.3, in the case of trust domains, separation of the authenticated principals from the end-points of a secure channel, and having security at the host-to-host datagram level, allows heterogeneity in implementation of secure channels and flexibility to change this implementation without the need to change any of the higher level protocols. Since higher-level protocols (e.g., transport-level) establish

associations between entities other than hosts (e.g., processes), if security mechanisms were placed at one of these layers, a trust domain would require an intermediate entity in the domain gateway to maintain separate higher-level associations (e.g., transport connections) with the entity within a domain and the entity outside the domain. In such a case, entities within a domain can never have secure connections directly with those outside and vice versa.

5.2. Disadvantages of transport layer security

Transport level protocols are used to implement a variety of communication paradigms. Request/reply (RPC) [4] and full duplex byte streams [21] are two of the popular communication paradigms. We examine secure RPC [4] as an instance of security in a RPC protocol and secure TCP [13] as an instance of security in a full duplex byte stream protocol. Both secure RPC and secure TCP are transport level protocols.

5.2.1. 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 sequence numbers of requests within a conversation. This converts simple stateless RPC into one with long-term state.
- A three-way handshake is necessary to agree upon the conversation key. The cost of this three-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 three-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 the encryption cost by piggybacking messages from different processes that are all destined to the same host.

5.2.2. Secure TCP

TCP is a DARPA Internet transport protocol [21] providing full duplex byte stream connections between processes on different hosts. Secure TCP [13] requires 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 additional disadvantages associated with this scheme:

- Four public-key operations are performed for each TCP connection.
- Encryption cost reduction by piggybacking is impossible since keys are not per host-pair.

5.3. Disadvantages of security above the transport layer

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

- Transport level protocols like TCP do connection establishment using three-way handshakes. If security mechanisms are above the transport layer, they require their

own handshake to agree upon keys after the transport level has established a connection. This duplication of handshaking entails higher message overhead.

- Transport level protocols do error detection using (insecure) checksums. Secure communication mechanisms above the transport layer must do their own cryptographic checksumming. This is an unnecessary duplication of effort as error detection at higher layers can be avoided if checksumming is done in the subtransport level.
- Transport levels employ sequencing to eliminate duplicates and out-of-sequence messages. Since an intruder could change the transport level headers and hence the transport level sequence numbers, security mechanisms above the transport layer must also do sequencing to detect such an intrusion, again resulting in 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 much unnecessary tearing down of connections and the associated performance overhead.
- Unauthenticated messages are detected only at the level where security mechanisms are. These messages are unnecessarily processed at all lower levels of the protocol hierarchy. Thus if the security mechanisms are at a high level, the amount of this unnecessary work can be large.
- Public-key operations cannot be reduced because authentication is not per-host, and single-key operations cannot be reduced by piggybacking.

6. Experimental Verification

A prototype of ADP has been implemented in C++ as part of the DASH Project in the Computer Systems Research Group at the University of California at Berkeley. This implementation runs on Sun 3/50 workstations connected by a 10 Mb/s Ethernet. Trace-driven experiments were performed using TCP and SUN NFS traces of network packets to verify some of the expected performance advantages over transport layer solutions for communication security. TCP was chosen as an example of a full-duplex byte stream protocol, and SUN NFS as an example of a request/response protocol. The primary performance indices were:

Latency L: the average delay incurred by a message between the instant it is given to the subtransport module for transmission and the instant it is delivered by another subtransport module to the destination process or to a higher-level protocol on the destination host. To compute L, we average the delays of the messages in a given finite sequence.

Throughput T: the maximum rate at which information can be transmitted by the subtransport layer and received by another subtransport layer on the destination host.

In a throughput experiment, the messages in the trace are transmitted at the maximum possible rate. In the transport-level case, when the arrival raate of a message trace is increased, a decision must be made about whether and how the process creation rate should be modified. There are two extreme cases: (1) The process creation rate is kept constant. This is one end of the spectrum and represents the best possible case for a transport layer that uses process-to-process secure channels. The throughput for this scenario will be denoted by T1. (2) The messages are assigned *a priori* to processes. Thus, when the message transmission rate is increased to its maximum value, the process creation rate

increases linearly. This represents the worst case for security at the transport layer. The throughput for this scenario will be denoted by T2. These two cases are of interest only for transport-layer security. For subtransport layer security the two cases will yield the same results.

Table 1 shows the latency and throughput values for the following cases: (1) security mechanisms in the subtransport layer (ADP), (2) security mechanisms in the TCP protocol, and (3) security mechanisms in the RPC protocol (NFS). The last two of these are instances of transport-layer security. From the table, we conclude that the performance gains of the subtransport-layer security over both instances of transport-layer security are substantial. The inferior performance of transport-layer approaches is to be attributed primarily to the much higher rate of channel establishment operations that these approaches require with respect to that caused by ADP. Channel establishment is quite time-consuming: we have measured in our system an average establishment time of 1.75 s.

Figure 4 shows the effect of piggybacking at the subtransport level. The input trace consisted of a sequence of messages leaving a busy file server. Among the many experiments we performed, an interesting one was intended to determine the variation of the latency as the arrival rate of messages in the input trace was progressively increased. Figure 3 shows that the effect of piggybacking is insignificant at low arrival rates. Without piggybacking, an increase in the message arrival rate causes a rapid increase in the latency. Table 2 shows the latencies and CPU idle times for an unmodified message trace (ALL) with an average message arrival rate of 250 messages/s, for the following cases: (1) without any communication security mechanisms, (2) without security mechanisms but with message piggybacking, (3) with subtransport-layer security but without message piggybacking, and (4) with subtransport-layer security and with message piggybacking. The difference in performance between cases 1 and 2 is considerable. The performance of case 4 is very close to that of case 2 whereas, the performance of case 3 is less than that of case 1. This shows that message piggybacking can keep the performance cost of encryption very small.

7. Conclusion

We have shown that, in communications between two parties whose processes run on kernels that are security-correct, it is possible to provide end-to-end authentication and privacy at the host-to-host (hence, subtransport) level. A constructive proof of this statement has been provided: we have indeed described a subtransport protocol, ADP, and proved that it can guarantee these end-to-end security properties.

We have also argued that the subtransport approach has a number of functional and performance advantages over the other, higher-level solutions. The performance benefits have been demonstrated by presenting some of the results of a comparison between ADP and a transport-level, process-to-process approach. Both the average latency of messages and the maximum throughput improve substantially when the security mechanisms are moved from the transport to the subtransport layer. These improvements are primarily due to the sharp decrease in the secure channel establishment rate.

Acknowledgement

Many individuals have contributed to the research described in this paper, most notably Gene Banman, Kevin Fall, Riccardo Gusella, Bob Lyon, Cherie Miller, Sechang Oh, Marty Rattner, Jean Richter, Bruno Sartirana, Brad Taylor, and Shin-Yuan Tzou. The authors wish to express their deep gratitude to all of them.

References

1. S. G. Akl, Digital Signatures: A Tutorial Survey, *IEEE Computer*, February 1983, 15-24.
2. S. G. Akl, On the Security of Compressed Encodings, *Advances in Cryptology: Proceedings of Crypto '83*, 209-230.
3. D. P. Anderson and P. V. Rangan, A Basis for Secure Communication in Large Distributed Systems, *Proc. IEEE Symp. on Security and Privacy*, Oakland, April 1987.
4. A. D. Birrell and B. J. Nelson, Implementing Remote Procedure Calls, *ACM Trans. Comput. Sys. 2*, 1 (Feb. 1984), 39-59.
5. A. D. Birrell, Secure Communication using Remote Procedure Calls, *ACM Trans. Comput. Sys. 3*, 1 (Feb. 1985), 1-14.
6. M. H. Cheheyl, M. Gasser, G. A. Huff and J. K. Millen, Verifying Security, *Computing Surveys 13*, 3 (Sept. 1981), 279-339.
7. D. R. Cheriton, VTMP: A Transport Protocol for the Next Generation of Communication Systems, *Proceedings of the Data Communications Symposium*, August 1986, 406-415.
8. D. E. Denning, Protecting Public Keys and Signature Keys, *IEEE Computer 16*, 2 (Feb. 1983), 27-35.
9. D. E. Denning, Cryptographic Checksums for Multilevel Database Security, *Proc. of the Symp. on Security and Privacy*, May 1984, 52-61.
10. D. E. Denning, Digital Signatures with RSA and Other Public-Key Cryptosystems, *Comm. of the ACM 27*, 4 (Apr. 1984), 388-392.
11. W. Diffie and M. Hellman, New Directions in Cryptography, *IEEE Trans. Information Theory IT-22*, 6 (Nov. 1976), 644-654.
12. W. Diffie, Conventional Versus Public Key Cryptosystems, in G. J. Simmons, ed., *Secure Communications and Asymmetric Cryptosystems*, Westview Press, Boulder, Colordo, 1982.
13. W. Diffie, Security for the DoD TCP, *Advances in Cryptology: Proc. of Crypto '85*, 1985, 208-227.
14. NBS, Data Encryption Standard, FIPS Publication 46, NBS, U.S. Dept. of Commerce, Washington, D.C., 1977.
15. R. L. Rivest, A. Shamir and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Comm. of the ACM 21*, 2 (Feb. 1978), 120-126.
16. J. H. Saltzer, D. P. Reed and D. D. Clark, End-to-End Arguments in System Design, *ACM Trans. Comput. Sys. 2*, 4 (Nov. 1984), 277-288.
17. A. S. Tanenbaum, Network Protocols, *Computing Surveys 13*, 4 (Dec. 1981), 453-489.
18. V. L. Voydock and S. T. Kent, Security Mechanisms in High-Level Network Protocols, *Computing Surveys 15*, 2 (June 1983), 135-171.
19. R. W. Watson, Timer-Based Mechanisms in Reliable Transport Protocol Connection Management, *Computer Networks 5*, 1 (Feb. 1981), 47-56.
20. RFC 791: Internet Protocol, Information Sciences Institute, University of Southern California, September 1981.
21. RFC 793: Transmission Control Protocol, Information Sciences Institute, University of Southern California, September 1981.

Table 1

Latency and throughputs of the TCP and NFS traces with the subtransport and transport approaches

	Latency (ms)		Throughput (o) T1 (kB/s)		Throughput (oo) T2 (kB/s)	
	TCP	NFS	TCP	NFS	TCP	NFS
Subtransport	6	8	90	325	90	325
Transport	30	42	76	305	18	152

(o) constant process creation rate

(oo) process creation rate linearly increasing with the arrival rate

Table 2

Latency and CPU overhead for the ALL trace

	Latency (ms)	CPU Overhead
No security mechanisms No piggybacking	67	39.9
No security mechanisms Piggybacking present	11	20.09
Subtransport security No piggybacking	107	59.79
Subtransport security Piggybacking present	11	20.3

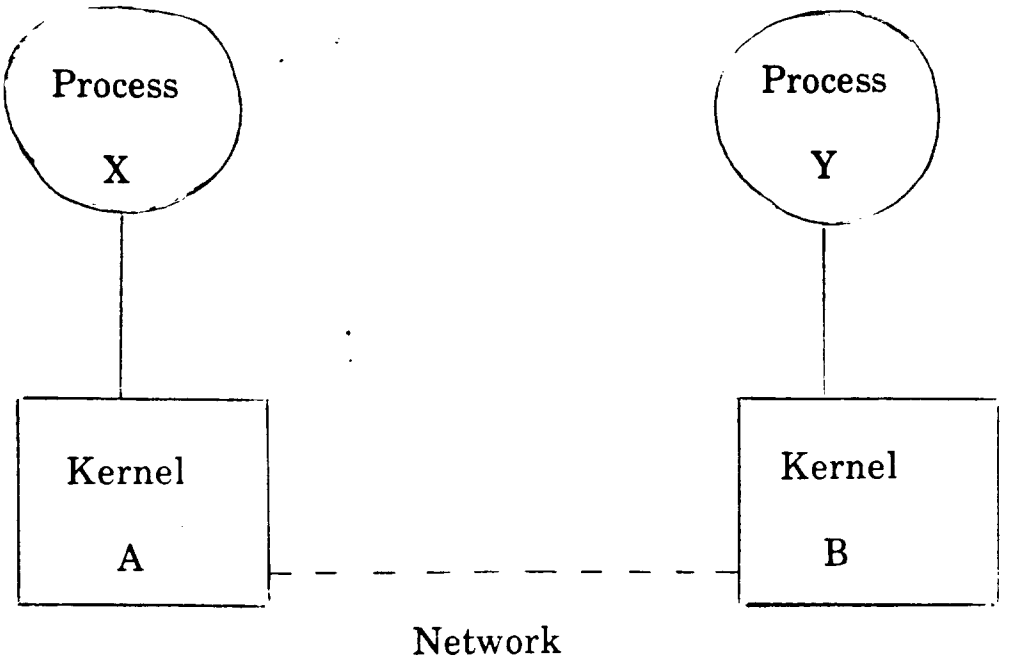


Figure 1.

Communication between two remote processes.

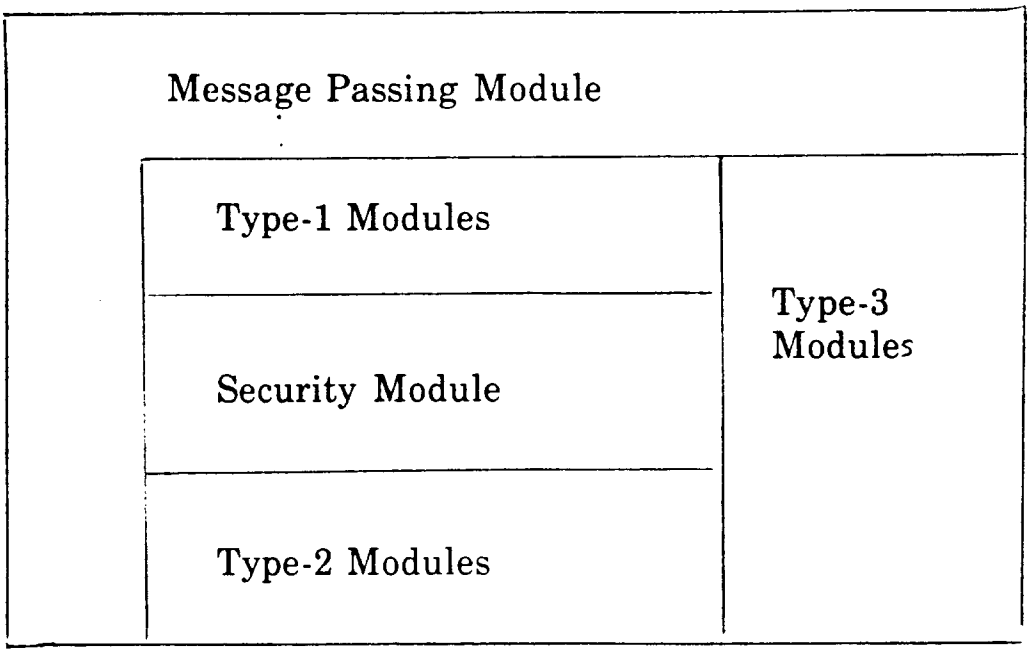


Figure 2.

Kernel Organization

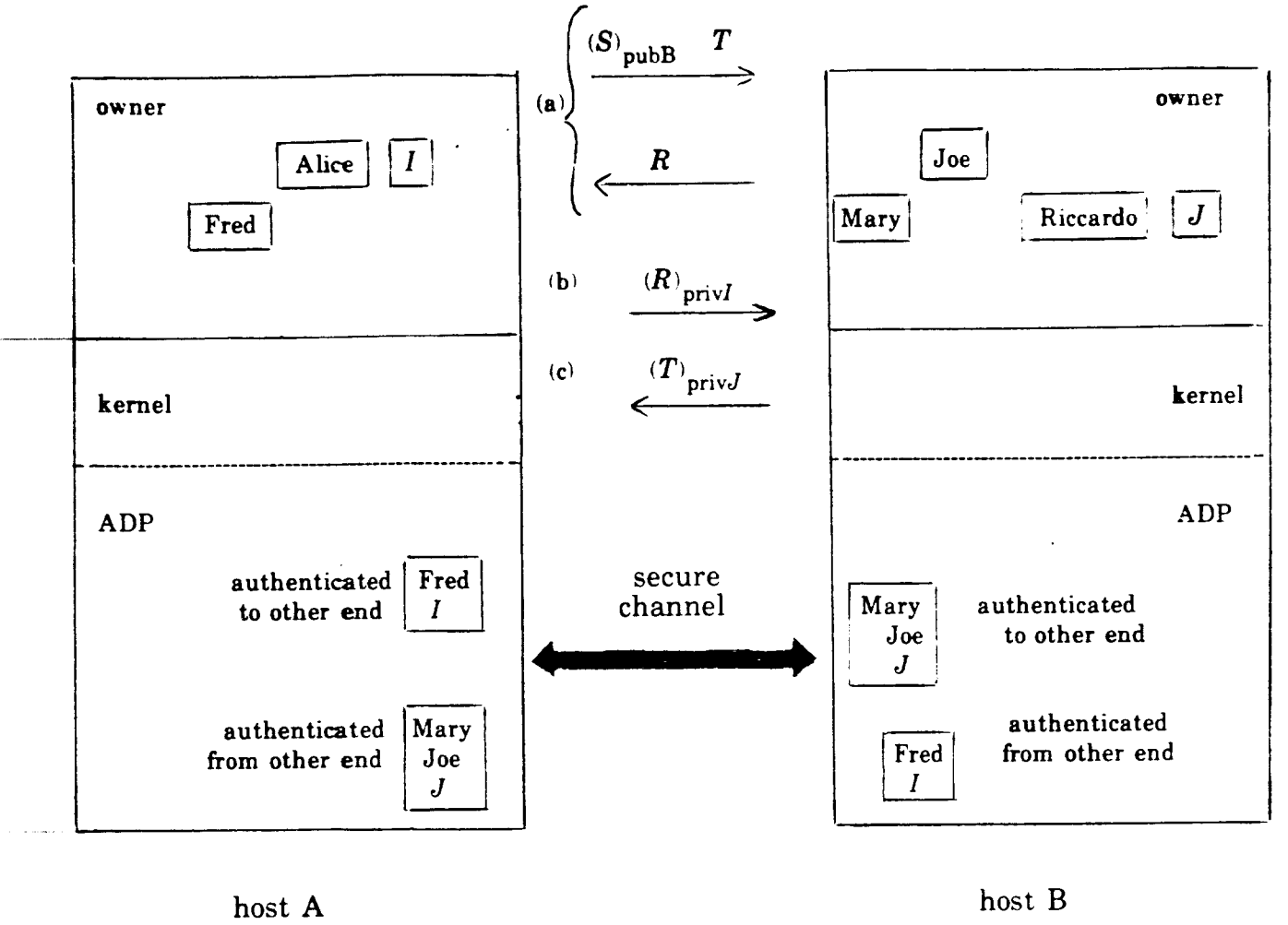


Figure 3.

A schematic diagram of ADP operation ((a) secure channel establishment; (b) authentication of owner I to (B); (c) authentication of owner J to (A). Messages exchanged in (b) and (c) as well as during normal communication between I and J are partially (or, optionally, totally) encrypted with channel key S. $(W)_z$ denotes string W encrypted with key z.

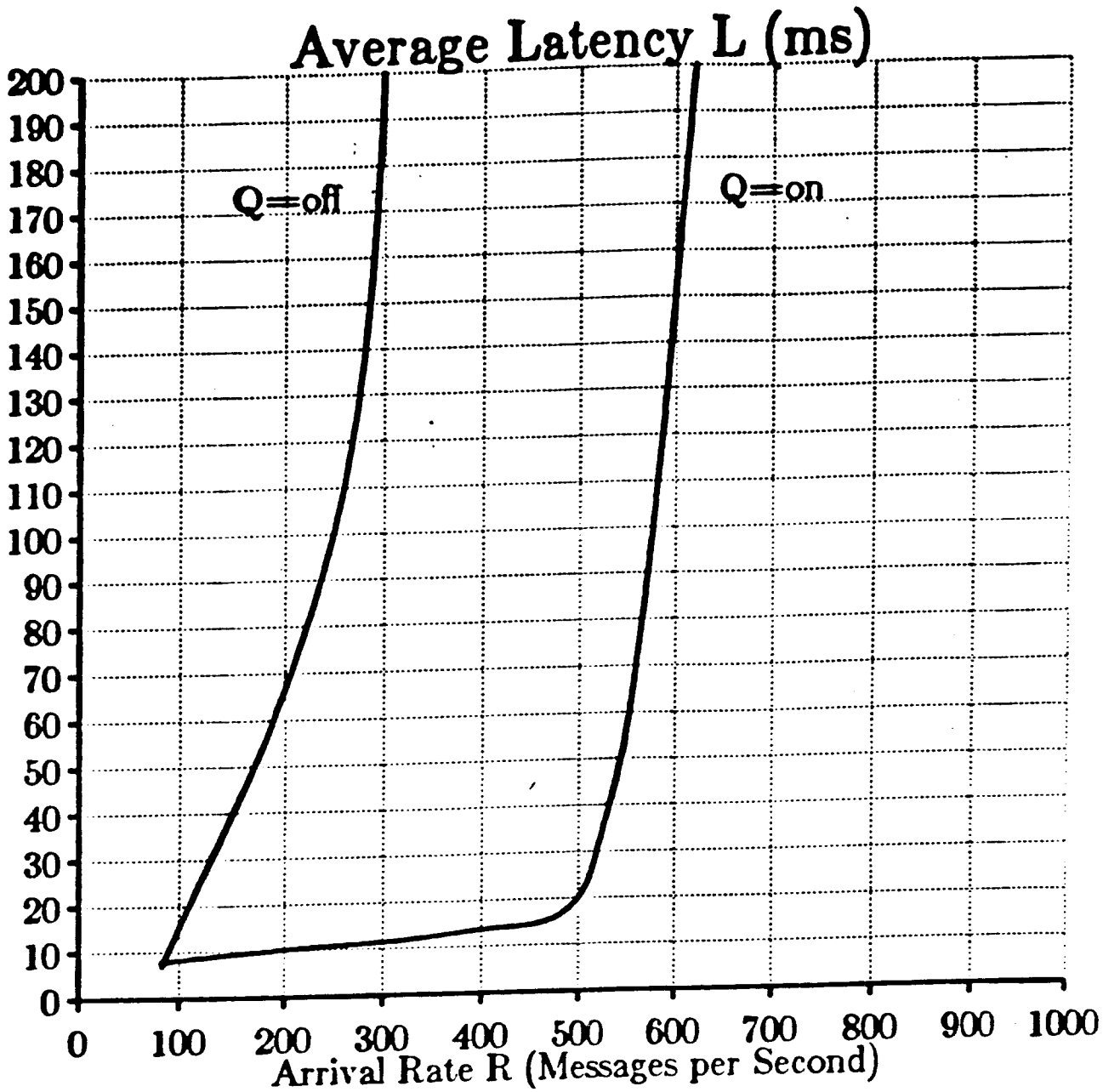


Figure 4.

Latency versus arrival rate for the ALL trace with piggybacking/queueing Q on and off.