Security for the DoD Transmission Control Protocol

Whitfield Diffie
Bell-Northern Research
Mountain View, California

1 *Introduction*

In securing packet switched digital communications, it is possible to add the security measures at almost any layer of the Open Systems Interconnection (OSI) model of network functioning. At one extreme, security may be supplied either by physical protection of the communication links (with no impact at all on network communication protocols) or by independent encryption of the traffic on each link of the network (with little protocol impact). Solutions of this sort are called *link security* and, although widely employed, have the disadvantage of requiring the users to place a high degree of trust in the network. At the other extreme, it is possible, using cryptography, to add security to each individual user level application. This has the advantage of minimizing the user's need to trust the network and thus providing *end-to-end security*, but also has the disadvantage of requiring a multiplicity of implementations.

A natural compromise is to attempt to place the security measures at the lowest point of full end-to-end communications, thereby achieving the benefits of end-to-end security with a single mechanism. As the provider of reliable end-to-end communications, the transport layer is the obvious choice for this location.

In this paper, we will pursue the transport layer approach by examining an existing transport protocol, the U. S. Department of Defense Transmission Control Protocol (TCP), and considering the ways in which this protocol could be made secure.

Our proposals will occur at three levels of compatibility starting with full compatibility with existing TCP and progressing through an upward compatible extension to the possibility of related but incompatible protocols.

2 *Overview of TCP*

This section provides an overview of the functioning of TCP and is largely drawn or paraphrased from the TCP specification[4]. As in that document, the abbreviation "TCP" will be used to denote both the protocol itself and programs used to implement that protocol.

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and especially in interconnected systems of such networks. It was explicitly designed for use with the DoD Internet protocol[3], but in principle, TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

2.1 *Facilities*

To provide its service on top of a less reliable "network" level communication system requires facilities in the following areas: Data Transfer, Reliability, Flow Control, Multiplexing, and Connection Management.

H.C. Williams (Ed.): Advances in Cryptology - CRYPTO '85, LNCS 218, pp. 108-127, 1986.

Data Transfer

The TCP is able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the network. In this stream mode, the TCPs decide when to block and forward data at their own convenience.

The sender can override the asynchronous character of the data transfer by setting the push flag in the TCP send command. This will make the sending TCP transmit all buffered data and set the push flag in the final resulting segment. The receiving TCP on seeing the push flag follows suit by forwarding all buffered data to its user.

TCP also provides a mechanism for communicating to the receiver of data that at some point further along in the data stream than the receiver is currently reading there are urgent data. TCP does not attempt to define specifically what the user should do upon being notified of pending urgent data, but the general notion is that the receiving process should take action to read through the urgent data quickly.

Reliability

Reliability is a complex issue that cannot be completely encompassed within a transport layer protocol. Redundant routing in the network, use of jam resistant communication links, and forward error correction all play a part in a comprehensive program of reliability.

Guarantees of reliability can be divided into two categories of which the second is a necessary building block of the first.

1) Assurance that the data will arive intact.
2) Assurance that the receiver will know whether the data have arrived intact or not.

A reliability mechanism providing some elements of each aspect is provided in TCP by the use of sequence numbers and acknowledgments (ACK's) to deliver data undamaged and in order at the destination. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first data octet in a segment is the sequence number transmitted with that segment and is called the segment sequence number. Each segment also carries an acknowledgment number which is the sequence number of the next data octet the receiver expects to arrive. When the TCP transmits a segment, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that segment is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted. At the receiver, the sequence numbers are used to order segments received out of turn and to eliminate duplicates.

TCP's acknowledgment and retransmit mechanism is augmented by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

It is important to note that an acknowledgment by TCP does not guarantee that the data have been delivered to the end user, but only that the receiving TCP has taken the responsibility for doing so.

Flow Control

TCP provides a means by which the receiver can govern the amount of information transmitted by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. This window specifies an allowed number of octets that the sender may transmit before receiving further permission.

Multiplexing and Connections

To allow many processes within a single host to use the communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. The concatenation of a port number with the host address from the network communication layer is called a socket.

The reliability and flow control mechanisms require that TCPs maintain certain status information for each data stream. This information, including sockets, sequence numbers, and window sizes, is called a connection and is uniquely specified by the pair of sockets it connects. A connection is defined by a pair of sockets, regardless of the processes plugged in to those sockets and TCP places no restrictions on a particular connection being used over and over again. Each new instance of a connection will be referred to as an incarnation of the connection. A local socket may participate simulaneously in connections to various foreign sockets and all connections are full duplex.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When communication is complete, the connection is terminated or closed to free the resources for other uses.

The binding of ports to processes is handled independently by each host. However, it is convenient to attach frequently used processes (e.g., a file server or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms in higher protocol layers.

Precedence and Security

In addition to the above features, TCP is also described as providing precedence and security. This, however, is security in the sense of computer operating system security and provides no protection in itself. It is only an option label passed through to the underlying network communication layer, which is expected to operate in a link secure environment. The security label is used by both the ends of the connection and any intermediate nodes to guarantee that classified segments will not be routed either to hosts with inadequate clearance or along paths with inadequate protection.

2.2 *The Host Environment*

The TCP specification assumes that TCP is a module in a computer operating system and that processes access the TCP much as they would access the file system. The TCP may call on other operating system functions to, for example, manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the network level datagram protocol module which may in turn call on the device driver. Despite this assumption the mechanisms of TCP do not preclude implementation of the TCP in a front-end processor, but in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described above.

In the environment of a verifyably secure operating system, implementation of TCP within the system itself would be perfectly acceptable from a security veiwpoint. In the absence of this

as yet unavailable technology, it is more desirable to isolate TCP together with the cryptographic machinery in a front end computer.

2.3 *TCP Interfaces*

The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/network layer interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls must have parameters for passing the address, type of service, precedence, security, and other control information.

2.4 *The Structure of the TCP Segment*

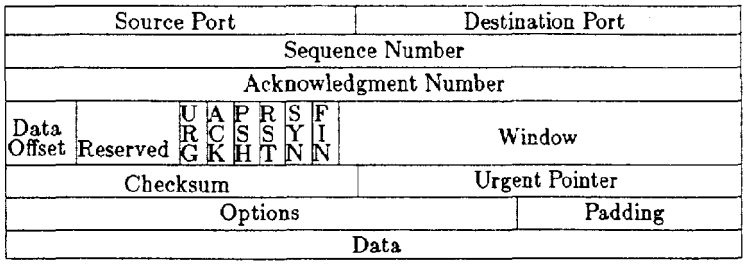


Figure 2.1 TCP Header Format

The TCP header block carries the sixteen bit names of the source and destination ports, but not the full socket names, which are carried in the underlying network layer datagram. It devotes thirty-two bits each to the sequence number of the first data octet in the segment and, if the ACK bit is set, to the value of the next sequence number the sender of the segment is expecting to receive.

A four bit data offset field specifies the length, in 32-bit words, of the TCP header. Six bits are reserved for possible use in future versions of TCP. Eight control bits explain the segment's purposes:

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

The 16-bit window field gives the number of octets beginning with the one acknowledged that the sender is currently willing to accept. The checksum field contains a checksum of the entire segment plus a pseudo-header containing data from the network layer. This checksum was designed for simplicity and makes no attempt to detect intentional tampering. If the URG bit is set, the urgent pointer contains the sequence number of the first octet following the urgent data.

The option field is of variable length and contains any selected options. Each option consists

of either one octet, for a fixed length option, or an option octet, an option length octet, and the option data. Following the options, the header is padded out to an integral number of 32-bit words.

2.5 *Establishing and Clearing Connections*

Since connections must be established between unreliable hosts and over a potentially unreliable communication network, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initiation of connections.

A connection, as mentioned earlier, may be opened and closed repeatedly by a variety of different processes. The problem that arises from this is how to identify duplicate segments from previous incarnations of the connection, a problem that is apparent if the connection is being closed and reopened in rapid succession, or if the connection is broken (with loss of memory) and later then reestablished.

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local connection name by which the user refers to the connection in subsequent calls. There are several things that TCP must remember about a connection and this information is stored in a data structure called a Transmission Control Block (TCB).

The OPEN call specifies whether connection establishment is to be actively pursued, or to be passively attended. A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket.

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The procedure used to establish a connection is called a *three-way handshake*. This procedure is normally initiated by one TCP and answered by another. This simplest three-way handshake is shown below. Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

```
    TCP A                                                TCP B
1.  CLOSED                                               LISTEN
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED
3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED
4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED
5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

Figure 2.2 Basic 3-Way Handshake for Connection Synchronization

The three way handshake also works if two TCP's initiate communication simultaneously.

```
    TCP A                                                TCP B
1.  CLOSED                                               CLOSED
2.  SYN-SENT     --> <SEQ=100><CTL=SYN>                  ...
3.  SYN-RECEIVED <-- <SEQ=300><CTL=SYN>                  <-- SYN-SENT
4.                ... <SEQ=100><CTL=SYN>                 --> SYN-RECEIVED
5.  SYN-RECEIVED --> <SEQ=101><ACK=301><CTL=ACK>         ...
6.  ESTABLISHED  <-- <SEQ=301><ACK=101><CTL=ACK>         <-- SYN-RECEIVED
7.                ... <SEQ=101><ACK=301><CTL=ACK>        --> ESTABLISHED
```

Figure 2.3 3-Way Handshake for Simultaneous Connection Synchronization

The examples above do not show connection synchronization using data-carrying segments, but this is perfectly legitimate, so long as the receiving TCP does not deliver any data to the user until it is clear the data are valid (i.e., until the connection reaches the ESTABLISHED state).

The clearing of a connection also involves the exchange of segments, in this case, segments carrying the FIN control flag.

3 *Meaning and Scope of Transport Layer Security*

In attempting to provide a secure transport layer protocol, we must answer two fundamental questions:

(1) What does it mean for communications in the transport layer to be secure?

(2) What does it mean for this security to have been applied by the transport layer?

The answer to the former question, as always, is that transport layer security is the combination of *privacy* (protection against disclosure of message contents to unauthorized parties) and *authentication* (a guarantee that the receiver knows the identity of the sender and that the message has arrived unaltered and without undue delay). In implementing secure transport protocols, however, it is valuable to refine this taxonomy.

As viewed from the transport layer, the opponent in an internetwork environment has the power not only to intercept, record, and examine all data passing over any connection, but to insert or delete messages at will. Privacy protection can be viewed as an attempt to limit the amount of information that the opponent can derive from these activities. Authentication measures are an attempt to assure that the opponents intrusions into the communication channel do not go unnoticed.

In the case of privacy, there is the possibility that even though an opponent is prevented from discovering the contents of any individual message, he is nonetheless able to make valuable deductions from an examination of the timing, length, and distribution of a variety of messages, a technique called *traffic analysis*. Protection for the contents of individual messages is called *message privacy*. Measures that prevent an opponent from studying the overall flow of communications are called *transmission security*.

Authentication is more complicated and is closely tied to the second question. In specifying that the receiver knows the identity of the sender, we must ask in what terms this identity is to be given. A transport protocol provides process to process communication, but these processes are known to the transport layer only through their association with sockets. A guarantee of the identity of the source of a message from the transport layer viewpoint is thus a guarantee that

segments actually emanate from some particular socket. This guarantee will be called protection against *unauthorized connection initiation.*

Given this limited view of the meaning of identity within the transport layer, it is reasonable to ask how socket identity is translated into the identities of entities in which trust is actually vested within the security system. This translation, however, takes place within higher level protocols that make use of the transport layer. The role of the latter is limited to supplying secure socket to socket connections.

The second criterion for the authenticity of data is a guarantee that messages have not been surreptitiously altered during transit; this guarantee is called assurance of *message integrity* or protection against *message stream modification.*

In either of the above cases it is also possible to distinguish different levels of quality in the evidence for authenticity. It is often the case that although the receiver of a message is able to assure himself that he knows the identity of the sender and the message has come through the channel unaltered, he would be unable to establish to a third party that he had not composed the message himself. If the reciever has the means of establishing the identity of the sender to the satisfaction of third parties, we say that the message bears a *digital signature.*

Some intruder actions may take the form, not of altering legitimate messages or even of sending new ones, but of delaying messages either for a limited period of time or indefinitely. The possibility that the intruder will delay messages sufficiently that their meanings have changed is called the threat of *replay.*

When the intruder goes one step further and delays messages indefinitely, the legitimate communicators are said to experience *denial of service.* This threat is often treated differently from others as it is often said that denial of service cannot be prevented, but only detected by authentication measures. A closer examination reveals that this is true of all threats to authentication. An intruder cannot be denied the chance to work mischief on the communication channel, but only prevented from doing it surreptitiously. In the case of message stream modification, however, countermeasures come so directly to the receiver's hand as to becloud the issue: A message that is recognizably inauthentic will be rejected immediately and the intruder will have achieved little. The practical effect of authentication is either to deter the intruder altogether or to convert all attacks into denial of service.

The use of protection against message stream modification opens the question of why false connections must be prevented at all. Data that come from illegitimate connections and data that started out from legitimate connections but were modified *en route* are, after all, indistinguishable to the receiver. Since each message must be authenticated before it is accepted, an unauthorized connection might be opened, but no useful data could be sent over it.

The answer lies in the second question. In saying that security has been supplied by the transport layer, we are saying that the higher level processes that appeal to the transport layer must be placing their faith in it, that the transport layer itself must be operating securely rather than merely serving as the conduit for secure communications. Any authentication procedure required to guarantee segment correctness must therefore be carried out by the TCP's. To limit authentication tests to the data alone and thus allow initiation of a connection (fail to check authenticity of SYN messages) even though no data from that connection would be accepted as authentic, serves only to leave an opening for the opponent to tie up the network with unauthorized connections.

4 *Securing TCP Cryptographically*

4.1 *Cryptography*

The basic approach to securing TCP will of course be to encrypt as much as possible of each TCP segment. In so doing, we need make only a few assumptions about the cryptographic system in use. These assumptions describe the operations of which it is capable[2,6], including whether it has the public key capability, but say nothing about its strength or internal functioning.

Public Key and Conventional Systems

In using cryptography to provide a secure transport service, either public key or conventional cryptosystems may be used. The advantage of the former are an improvement in the security of key distribution[2] and the availability of digital signatures. The latter have the advantage, at least for the present, of both higher performance and greater familiarity.

A public key system can perform all of the tasks of a conventional system, even though in some of these it can make no use of its public key capability. A single, public key, cryptosystem might therefore be used for all encryption within a network. At present, however, the low speeds and large block sizes of public key systems make them undesirable for any application in which their special capabilities are not required and a combination of public key and conventional systems is the most satisfactory arrangement.

It is also possible to operate a conventional cryptographic system in the style of a public key system, thereby minimizing the effect on protocol structure of the decision to select one or the other. The user of a public key system employs one key (the other user's public key) for sending messages and another (his own private key) for receiving them. The same approach can be adopted in the conventional case with each user employing one key to encrypt his outgoing messages and another to decrypt the incoming ones.

It is important to remember that a conventional system operating in the public key style does not provide the public key functions; both keys must be treated as secret and no message can be regarded as digitally signed. It is equally important to note that this has little effect within the transport layer. At higher layers the distinction between conventional and public key systems affects the form of the protocols; in the transport layer, it affects only the quality of the protection provided.

Cryptographic systems in the rest of this paper will always be described in the public key form. Each party will have both a sending key (other party's "public key") and a receiving key (his own "private key"). It is convenient for the lengths of keys to be powers of two. Keys for conventional systems are typically between 64 and 256 bits in length while public keys are at present somewhat longer, running from 256 bits up to about a thousand.

Modes of Operation

All cryptographic systems to be used are assumed to be capable of operating in one of the following modes:

(1) *Cipher block chaining* mode (of which *block mode* is a special case) with blocklength n.

(2) *Cipher feedback* mode on chunks of text of any size not longer than n.

(3) Synchronous modes such as *counter driven* mode or *Output Feedback* mode on chunks of text of any size not longer than *n*.

The most common forms of cipher feedback operate on either a single bit at a time or on eight bits at a time. Because of the octet oriented structure of TCP, eight bit cipher feedback is the most natural choice for encrypting TCP segments. In cipher feedback mode, however, a system can make no use of the public key property and cipher block chaining might therefore be selected for this purpose.

Synchronous modes of cryptographic operation have been popular in communication systems because they do not propagate errors and thus offer good performance in the presence of noise. This feature has no direct effect on TCP itself and is generally less applicable at the transport level of packet switched networks because of error correction at lower levels. Nonetheless, there would be no disadvantage in using synchronous modes with TCP and this might in some cases provide a convenient compatibility with existing equipment.

Message Indicators and Cryptographic Checksums

It is preferable for TCP segments to be independently decryptable, since the alternative requires that sufficient information be left in clear to allow segment ordering before decryption. The cost of this decision is additional information in each segment, telling the receiver the cryptographic state in which to begin decrypting the message. This information is variously called a *message indicator* or *initialization vector* and should, for security's sake, be no less than 64 bits in length.

In both the cipher block chaining and cipher feedback modes of encryption, each item of text is encrypted or decrypted in a manner that depends not only on the key, but on some quantity of the preceeding cipher text. In these modes, the message indicator plays the role of this quantity.

For authentication purposes, the cryptographic system must be capable of generating a cryptographic checksum for each segment transmitted. This checksum is of the order of 64 bits in length and depends on three different types of data:

(1) Data included in the segment in encrypted form.

(2) Data included in the segment, but not encrypted.

(3) Data associated with the segment, but already known to the receiver and not transmitted.

When a public key cryptosystem is employed, the cryptographic checksum can play the role of a digital signature. This can be accomplished either by applying the public key system directly to all of the data to be signed, or by computing a cryptographic checksum with a conventional system and then signing the checksum.

Key Management

Since key distribution is a process that is handled primarily above the transport layer, it will not be examined in detail here. For our purposes, it will be sufficient to assume that when a TCP connection is opened, keys specific to that connection have already been placed in position at its ends.

4.2 *Message Privacy*

Message Privacy is accomplished by encrypting all data in the TCP segment and as much of the header information as possible. The amount of header information that can be protected depends on the degree of compatibility that must be maintained between secure and unsecured TCP. If full compatibility (interoperability with existing TCP implementations) is required, all header data must be left in clear. On the other hand, in a network where all TCPs incorporate security and all segments are required to be encrypted, encryption can be extended to the header as a whole. In a network where both secured and unsecured TCP connections are permitted, some means must be provided for distinguishing between encrypted and unencrypted segments.

Unlike the other elements of the header, the source and destination ports present a particularly difficult problem with respect to encryption. Since connections occur between pairs of ports, port numbers are just the lowest order part of the packet address and from this point of view should merely be passed in clear. This, however, although convenient, is undesirable and probably unnecessary. It is undesirable because the port numbers provide information of great value to a traffic analyst. It is unnecessary since the port numbers (unlike other parts of the address) distinguish between processes all of which are located within a single physically secured location.

If the source and destination ports were encrypted in the connection specific keys, the receiving TCP would have no way of discovering for which of its ports an incoming segment was intended. Its only hope would be to decrypt the segment under the key associated with each possible port. Although procedures of this kind are suitable in some cases, the process would be too time consuming to be applied to each incoming segment.

In order to avoid leaving the port numbers in clear there are two possibilities. Either all segments must be encrypted with a key associated with the host pair rather than the process pair or the port numbers alone must be encrypted with such a key. It appears preferable to encrypt only the port fields using host pair keys, since associating keys solely with host pairs appears to present the same difficulties as having an additional host pair key and is made awkward by the fact that communication is synchronized on a connection basis.

Host pair keys must either be provided by the key distribution mechanism along with the session keys or derived therefrom by the communicating TCP's. Any scheme presents some bookkeeping problems: When a second connection is opened between the same pair of hosts, the corresponding TCP's must cooperate in either maintaining the first host pair key or (probably better, but more difficult) switching to the second. This task cannot be borne by the Key Distribution Center unless there is only one KDC in use by the two hosts and this KDC is required to maintain awareness of all connections in progress.

4.3 *Transmission Security*

Even when the whole segment, including the entire header, is encrypted, the lengths, timings, and host addresses of segments will be visible to traffic analysts. This is a problem that is not readily attacked in transport layer protocols.

The essence of transmission security is concealing traffic patterns from an opponent by sending dummy messages. The role of cryptography in this process is vital but limited: it prevents the opponent from distinguishing real messages from dummies.

Transmission security measures can readily be applied at the link level in circumstances where

the cost of communication does not depend on the volume of traffic. In this case, the link is kept constantly busy with a stream of encrypted data whether there are real messages to send or not.

It is also possible to apply transmission security measures in the network layer of broadcast networks. Under these circumstances, the origins and destinations of messages can be concealed by using cryptography as the addressing mechanism. All messages are encrypted and a station recognizes the messages addressed to it by finding that it can decrypt them successfully. The existence and lengths of messages are harder to conceal. Dummy messages can be sent at little intrinsic cost, but they must be managed very carefully to avoid congesting the network.

Applying transmission security on an end-to-end basis in point to point networks (where addressing information is needed by the intermediate nodes) is extremely difficult and may properly belong to the network layer rather than the transport layer. In order to conceal all traffic flow information, messages must be transported by *flood routing*: The point to point network mimics a broadcast network by routing all messages to all possible addresses. This, of course, is feasible in only the most exceptional circumstances. It may, however, be possible to increase the traffic analyst's burden's, at acceptable costs, by sending only a moderate number of additional messages, particularly if resources are dedicated to relaying messages[1].

4.4 *Secure Connection Management*

Security gives meaning to the concept of connection above and beyond that already present in TCP. A secure connection is the fundamental service provided by a secure transport protocol and is characterized by the use of a particular set of cryptographic keys. In this light, the TCP concepts of "connection" and "connection instance" deserve further examination.

In TCP a connection is defined entirely by a pair of sockets. Intuitively, this concept is overbroad, failing as it does to distinguish between two quite different cases. In the former, two processes, each of which owns a particular port, repeatedly open and close the TCP connection between them. In the latter, a connection is opened used and closed by a pair of processes, then at a later time the same pair of sockets, hence in TCP terms the same connection, is employed by a new and unrelated pair of processes. For TCP's purposes, these cases are indistinguishable; its concern is solely to be able to to discern and reject segments that are not intended for the current incarnation.

We will use the term *session* to distinguish connections of the former type, connections unified by the use of a single set of cryptographic keys. This term reflects the fact that these connections are arranged by key distribution protocols operating in the session layer of network architecture.

It is still necessary to be able to distinguish one incarnation of the connection from the next and so we will further distinguish between *session keys* and *incarnation keys*. The former are supplied by a higher level key distribution mechanism, while the later are arranged locally by the corresponding TCP's in the course of opening the connection.

It is also possible to take the more restrictive view that each incarnation of a connection is a distinct entity and thus to require old keys to be discarded each time a connection is closed and new keys to be distributed each time a connection is opened[5]. We will adopt the viewpoint above as being closer to that of unsecured TCP. Note however, that while a closed unsecured TCP connection leaves no trace in the participating TCP's, a closed secure connection requires a key to be preserved for later use either by TCP or by some closely associated mechanism.

In many applications a secure connection will be limited to a single incarnation. The procedure in this case, however, is the same as for the multi-incarnation case: an incarnation key is derived from the connection key during the open.

Establishing a Secure Connection

Secure connection initiation requires the addition of a challenge and response authentication procedure to the three way handshake. Before connection initiation can begin, keys must have been delivered to the corresponding TCP's. In the fundamental case, where one process is initiating a connection to another, the key distribution mechanism will also provide each TCP with a specification of both sockets. Once this has occurred, the "calling" TCP begins an active open sequence and the "called" TCP begins a passive open. The basic structure of such an exchange follows; the issue of how the data are incorporated in TCP segments depends on the degree of compatibility with unsecured TCP that is required and is discussed in later sections. As described earlier, all keys will be presented in pairs in the public key form.

The calling TCP, A, constructs a SYN segment to which a challenge has been added.

TCP-A→TCP-B: $\{A\text{'s challenge}\}^{B\text{'s public key}}$

When the called TCP, B, receives this segment it returns a SYN packet to A both answering A's challenge and presenting a challenge of its own.

TCP-B→TCP-A: $\left\{B\text{'s response} = \{A\text{'s challenge}\}^{B\text{'s private key}}, B\text{'s challenge}\right\}^{A\text{'s public key}}$

Using TCP B's public key, TCP A can decrypt the first half of this message to verify that its challenge has been answered correctly and can recognize the latter part as a challenge to which it must respond.

TCP-A→TCP-B: $\{B\text{'s challenge}\}^{A\text{'s private key}}$

This exchange, in addition to unsecured TCP's synchronization of sequence numbers, demonstrates to each party that the other is the party to which it had been referred by the key distribution mechanism. At the same time its serves to exchange pieces of information (the challenges) that will serve as the keys for the current incarnation.

The requirement in unsecured TCP that sequence numbers be generated in a non-repeating, clock dependent, manner is replaced by a similar but more exacting mechanism for generating the incarnation key This allows the sequence numbers themselves always. to begin at zero, since segments from previous incarnations can be recognized as being encrypted under outdated incarnation keys.

TCP allows the possibility that both ends of a connection may attempt to open simultaneously. Although this is unlikely at the beginning of a secure connection, it can occur when a connection is reincarnated. The only effect of independent opens is that the second message above will appear as two distinct messages rather than one combined message.

Passive Open with Unspecified Foreign Socket

The secure connections discussed above all take place between fully specified socket pairs; unsecured TCP, however, allows the possibility of a passive open with an unspecified foreign

socket. Among secure TCPs, this event can only occur in limited cases, but these cases are very important. This mechanism both impinges on the domain of key distribution and is required by the key distribution center.

When a listening TCP receives an encrypted SYN from an unpredetermined foreign socket, it must determine what key to use. This problem is dramatically simplified if the connection keys are public keys, in which case the session key will be a public key for the listening process regardless of the identity of the calling process. Using conventional keys, however, this determination is more complex.

If the calling process is always assigned the same port by its local TCP, then the listening TCP can determine the correct key from the foreign socket. This is probably the most general possibility that can be allowed since in any other case the contacts, although between the same two processes, are not in the TCP sense the same connection.

Message Integrity

Message integrity is gained by adding a cryptographic checksum (also about 64 bits in length) to the segment. This checksum covers the pseudo-header, header, and data and must be correct in order for a segment to be acknowledged.

This mechanism also extends the sender identification established during connection initiation by demonstrating that the sender of the segment knows the incarnation key that was agreed on during the challenge and response.

4.5 *Detection of Replay*

Unsecured TCP provides a mechanism for recognizing and rejecting segments from previous incarnations that have been lost long enough in the network to be mistaken for segments from the current incarnation. Secure TCP must reject in addition segments held for arbitrarily long periods, and subsequently replayed, by an opponent. Fortunately, the cryptographic techniques used to provide security also provide a simpler and more reliable means of making this distinction.

There are two fundamental means for judging the timeliness of messages. If the sender and receiver have synchronized clocks, a message whose integrity is guaranteed can also be authenticated as timely by examination of an included time field. This time field can be expressed either in hours minutes and seconds or, as with TCP's sequence numbers, in terms of the amount of data sent and received. If synchronized clocks are not available timeliness can be verified by a challenge and response procedure. Data will be recognized as current if they are tied to the response to a current challenge.

Secured TCP uses both of these mechanisms. The incorporation of a challenge and response procedure in initiating the connection guarantees the timeliness of the SYN segments. These in turn serve to synchronize a clock (the sequence numbers) in terms of which the timeliness of all later segments is verified.

Replayed segments will be detected as inauthentic either because they come from earlier incarnations of the connection, and are thus encrypted in the wrong incarnation key, or because they come from earlier in the same incarnation, and thus have the wrong sequence number.

4.6 *Detecting Denial of Service*

Some protection against denial of service is built in to TCP through the acknowledgment mechanism: a sending TCP cannot remain unaware that a segment has failed to reach its destination. A TCP that is not transmitting, however, but merely waiting for a message from the other end of the connection has no way of knowing if this message has been blocked or merely has yet to be transmitted.

To counter this possibility, a TCP that has not received a segment in some time can challenge the other end of the connection to demonstrate its availability and authenticity. This exchange is similar to that used to initiate an incarnation and can be used to change the incarnation key at unpredictable times during the session. This procedure can be used not only to detect denial of service, but to counter subtle vulnerabilities that make the use of a public key system to exchange conventional incarnation keys less secure than the use of public keys throughout[2].

5 *Full Compatibility—An Added Layer of Protocol*

Full compatibility with TCP does not permit encryption of the header or even any changes or additions thereto. Any action that is to be taken must consist solely of additions to and encryption of the user data. These additions, furthermore, must take place prior to, and therefore in ignorance of the actions of TCP. This renders such otherwise plausible acts as adding a cryptographic checksum of the entire TCP segment infeasible.

The effect is of the addition of an extra layer of protocol at the upper edge of the transport layer. A host adopting this approach will present to the world an entirely correct TCP appearance and may even have unsecured conversations with standard TCPs. A process requiring a secure connection, however, must make use of TCP not directly but through the added security layer.

Under these circumstances some of TCP's functions are difficult or impossible to duplicate without building almost full transport layer functioning into the added security layer.

5.1 *Data Privacy and Authentication*

From TCP's point of view, the security layer is a user process and TCP will therefore hand it data that TCP believes to be damage free and in proper order. This allows the security layer to protect the privacy of the data by encrypting them as a single *cipher chain* and frees the security layer from the need to add a message indicator to each segment. The result is a reduction in overhead when transmitting ordinary data.

One price that is paid for this reduction in the normal case is increased overhead in transmitting urgent data. When the security layer hands TCP a buffer with the urgent flag set, it must incorporate a message indicator as the first element. When the security layer receives urgent data from TCP, it must treat the first portion as a message indicator and decrypt the data accordingly.

For authentication, the security layer must also compute cryptographic checksums on both the data and certain information from the TCP header and pseudo header. These are added by the sending security layer and used by the receiving one to test for alteration. In order to verify timeliness, these checksums must cover some equivalent of the sequence number; in order to detect segments maliciously routed back to the sender, they may need to cover the full socket pair.

Fortunately, although the elements of the header generated by TCP are not available to the security layer, the information given to TCP in the OPEN or SEND calls is. Requests to open or

use a secure connection are made to the security layer and passed thereby to TCP; this gives the security layer access to both socket addresses, although not the sequence number, acknowledgment, or window. To replace the sequence number to which it lacks access, the security layer generates a sequence number of its own by counting octets.

For the sake of generality, it is desirable that authentication be accomplished without imposing any structure on the data beyond the segmentation carried out by TCP itself. This, however, presents a difficult problem. On transmission, the cryptographic checksums can be attached to the data given to TCP by the security layer. Recognizing these checksums in the received data is another matter. The security layer is not only unable to get TCP header information, but on receiving, it is unable to distinguish the data comprising individual segments.

In order to make the segment structure of received segments visible to the security layer markers must be placed in the data stream where they will be passed through by TCP. Since TCP provides a transparent channel, any code reserved for this purpose can be expected to make occasional independent appearances in the data. Preventing such patterns from causing disruption requires use of bit or character stuffing techniques to alter the reserved pattern whenever it is seen by the sending security later.

In the event that inauthentic data are detected by the receiving security layer, its options are quite limited. It has no meaningful way to reject the segment, which has already been accepted by TCP, unless it duplicates the entire acknowledgment and retransmission mechanism. On the other hand, data that have already been accepted by TCP yet are found to be inauthentic must have been intentionally manipulated and the security layer can reasonably respond by sending an alarm message back to the user process and a reset to the TCP connection.

5.2 *Connection Initiation*

Use of a separate security layer means that the triple handshake of TCP must be completed and then mirrored in a similar cryptographic handshaking procedure by the security layer. At first glance, it would appear that these two processes could be at least partially combined by making use of the data carrying abilities of TCP SYN segments. This approach fails, however, because TCP will not deliver the data in these segments to its user (the security layer) until its own connection setup process is complete.

5.3 *Detecting Denial of Service*

The separation of the security layer from TCP deprives the former of the level of denial of service protection supplied by TCP. The security layer can check arriving data for integrity, but would require a full acknowledgment mechanism of its own to be sure that data it sent had arrived. Implementation of more refined denial of service detection, however, is straightforward and mimics the initial authentication exchange.

6 *An Upward Compatible Extension of TCP*

An upward compatible extension of TCP simplifies the introduction of security by allowing the security mechanism direct access to the structure of TCP segments. This permits the segment as a whole, including almost all of the header information to be encrypted. The exceptions are a bit indicating whether or not the segment is encrypted and perhaps the local and foreign port addresses.

The problem of encrypting the port addresses has been touched on in an earlier section. They

cannot be encrypted in a connection specific key since they are used precisely to distinguish between the various possible connections. They can, however, be encrypted in a host pair key. This has little effect on the principal functions of TCP and presents primarily a problem of installing and changing these keys at suitable times.

In most cases, the receiving TCP does not need to distinguish between encrypted and unencrypted segments because it will have been informed by the key distribution mechanism that an encrypted connection is to be created between specified local and foreign sockets. Aside from the possibility that encrypted and unencrypted segments may be allowed in the same session, the principal circumstance in which TCP might be required to distinguish is that of a passive open with unspecified foreign socket.

Since the receiving TCP must be able to read the "encrypted" bit before decrypting the segment, this bit must be located in a fixed position relative to the segment's beginning, a constraint that precludes the use of the option area. The best solution appears to be using one of the reserved bits to indicate an encrypted segment. This assumes that the receiving TCP has only one cryptographic system at its disposal or that it has been informed by some other means of which system to use. This, however, is a natural assumption, since any other approach would be open to criticism on transmission security grounds.

Encrypting the whole segment requires that decryption be carried out before segment reordering and therefore that each segment must be independently decryptable; this in turn mandates the addition of a message indicator to each segment. Since the receiving TCP cannot decrypt the segment correctly unless it is able to locate the message indicator, this item, like the "encrypted" bit must occur in a fixed position, even though it only occurs in encrypted segments.

One more item must be added to the segment: a cryptographic checksum. Unlike the message indicator, this need not be locatable prior to decrypting the segment. This allows greater freedom in its placement, but it seems clean and convenient to put it directly after the message indicator.

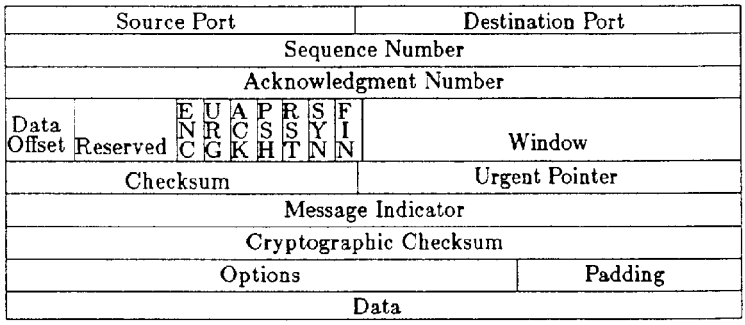


Figure 6.4 Encrypted Segment Header

The checksum occurs in every encrypted segment and is calculated from the entirety of the pseudo header, the header, and the data with the exception of the checksum itself, which may either be omitted from the calculation or replaced by zeros. In encrypted segments, the function of the 16 bit, non-cryptographic checksum in determining segment acceptability is supplanted by use of the cryptographic checksum.

In segments with the SYN flag set, additional data must be incorporated for the challenge and response aspect of the handshaking. These are probably best included in option fields, and we will add the two options CHAL and RESP. This permits these options to be used by themselves for various reinitializations of the incarnation key.

6.1 *Privacy, Reliability, and Authenticity*

Except during the opening of a connection, TCP's many functions operate protected, but barely affected, by encryption. An opponent examining intercepted segments can observe that their encrypted bits are on, and can confirm this observation by attempting to read the data, but can observe nothing more than the total length of the segment. The connection to which the segment belongs, the segment's data, and the various fields that would reveal how many octets have been sent, how many acknowledged, and whether the segment is a SYN, FIN, or retransmission are all concealed.

The acknowledgment and retransmission mechanisms of secured and unsecured TCP operate in exactly the same way except that the cryptographic checksum replaces the non-cryptographic in deciding whether to accept a segment. An opponent can neither alter the subscribers' data nor affect the connection's behavior by inserting phony control segments, since any segment received is first judged for authenticity by its checksum and then for timeliness by its sequence number.

6.2 *Secure Connection Initiation*

The unsecured three way handshake assures both participating TCP's of the timeliness of the connection and allows them to reject stray segments from previous incarnations with high reliability. Each side chooses an initial sequence number for the current incarnation, sends this sequence number to the other TCP and recieves in return an acknowledgment of this choice. Each TCP must both have acknowledged the other's starting sequence number and received an acknowledgment of its own before it will regard the connection as open and accept data for passage to its user.

The initiation of secure TCP connections follows this pattern in form and extends it in objectives, verifying the timeliness of the connection as well as the more fundamental fact that the participating parties share a compatible set of cryptographic keys.

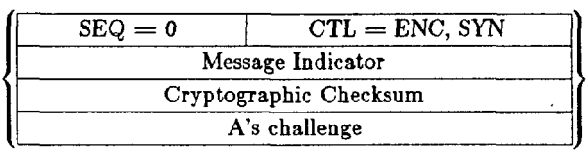
Rather than agreeing on initial sequence numbers, secure TCP's agree on a set of keys for use in the current incarnation. This allows segments from previous incarnations to be detected not on the basis of bad sequence numbers, but on the inability of the receiver to decrypt them and derive a correct cryptographic checksum. This procedure is less prone to accidental failure than the unsecured version since keys are never less than twice as long as TCP's 32-bit sequence numbers and accidental repetitions are correspondingly less likely. It also frees the participants from the need to select random starting points in the sequence number space and allows both to begin at zero.

In a secure connection, sequence numbers can never be permitted to cycle during the use of a single key since this would not allow new segments to be distinguished from old (played back) segments with the same sequence number. In fact, keys are not expected to remain in use for nearly this long, but rather are changed periodically by a mechanism to be discussed in connection with detecting denial of service.

In the most common case of secure connection initiation, one end of the connection, which we

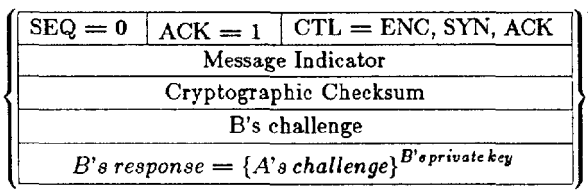
will denote as A, starts proceedings with an active OPEN while the other, B, makes itself receptive to an arriving segment by doing a passive OPEN.

A→B:



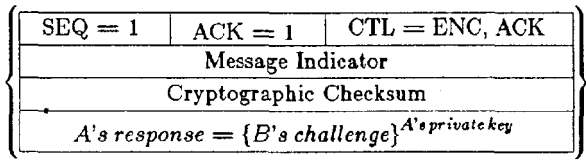
B responds by answering A's challenge and posing one of its own:

B→A:



A is now able to verify that B has correctly answered its challenge by decrypting B's answer with B's public key. The cryptographic version of the three way handshake is concluded by A's answer to B's challenge:

A→B:



Once B has checked A's signature on the challenge, both processes are in an ESTABLISHED state and are willing to accept and acknowledge data. As with an unsecured handshake, the synchronization messages can carry data, but these data must not be accepted and passed on to the user until the handshake is complete.

Throughout connection initiation, the segments exchanged are encrypted in the correspondents' session keys. Once the TCP's enter the ESTABLISHED state, however, they will switch to using an incarnation key, manufactured from the exchanged challenges, for the duration of the incarnation. There are various ways to produce such an incarnation key, but we will adopt the convention that the challenges are treated as exchanged public keys, regardless of whether a public key or conventional system is in use. Each side of the connection will thus transmit in one key and receive in another.

The switch from the session keys to the incarnation keys opens the possibility of doubt on the receiving TCP's part about which key to use in decrypting an incoming segment. Once an incarnation key has been selected, this will become the key of choice and most segments otherwise encrypted will represent errors. If, however, a segment fails to decrypt correctly using the incarnation key, the session key can be tried.

6.3 *Detecting Denial of Service*

Denial of service can be reliably detected by the sender of messages, say A, through its failure to receive acknowledgments. After a number of attempts that will vary with circumstances, it will

respond by sending a trouble report to its user. An intruder cannot defeat this strategy by sending phony acknowledgments because he is unable to make his phonies cryptographically acceptable.

The prospective receiver, B, on the other hand has no way of knowing that segments intended for him are being prevented from reaching their goal. If B is to discover this, he must send messages to A that will provoke a response.

The same challenge and response mechanism used in establishing the connection is suitable for this purpose. Either party may at any time during the connection, whether it feels deprived of incoming data or not, send a new "public key" to the other party and expect a satisfactory response. This challenge can be given a segment to itself or combined with user data. This latter possibility helps to prevent an opponent from distinguishing such messages from data and allowing only the former to pass.

It is interesting to note that challenges posed in this manner arrive in segments encrypted with the current incarnation key, but must be signed with the receivers private key. A correct response therefore guarantees that the responder knows both.

After a key change, arrival of legitimate data encrypted with the old key is not unlikely and the receiver must be prepared to hold it until all segments sent before the key change have been received and acknowledged. The sender is under no such obligation and can freely retransmit unacknowledged segments in the new key rather than the old.

As noted earlier, this mechanism not only serves to detect denial of service but to prevent recycling of sequence numbers.

7 *Incompatible, But Related, Protocols*

TCP is best suited to establishing connections across which substantial amounts of data will be transferred asynchronously in both directions; it is inefficient for transmission of small amounts of data such as remote procedure calls and, due to its insistence that every segment must be received undamaged and acknowledged, ill suited to carrying real time data such as voice.

TCP's inefficiency in transmitting small amounts of data has a direct effect on its suitability for communication between a KDC and its clients since key exchange requires only very short messages and, unless each client maintains a constant connection with the KDC, TCP will add substantial overhead. The specialization of TCP has a more profound effect, however, on its utility as the common denominator of secure communications and the primary location for the network security mechanism. This utility is dependent on the assumption that all processes above the transport layer will make use of TCP and can thus relay on it to provide security. If instead, there must be several secure transport layer protocols, not only must security be incorporated in all of them, but each must be provided with access to cryptographic hardware.

TCP segments are best viewed as "programs that are mostly data" and TCP as an interpreter for executing these programs. In this view, an effort to make TCP more flexible would probably change it from a language with a nearly fixed length instruction set to one with a variable length instruction set. Instead of requiring that all of the various fields: acknowledgment number, checksum, window, etc. be present in every segment, the segment header would begin with the control bits and incorporate additional fields as needed.

ENC	URG	ACK	PSH	RST	SYN	FIN	ECN	CWR	Reserved	Data Offset
Source Port									Destination Port	
Sequence Number										
Acknowledgment, Window, etc. (as needed)										
Options and Padding										
Data										

Figure 7.5 Possible Alternative Header Format

In a configuration analogous to that presented in the previous section, a security control bit would be set. This would indicate that the segment must be decrypted and that the acceptance test would be cryptographic. In this case a checksum control bit would not be present and no non-cryptographic checksum would be performed. The present TCP layout in these respects would be modeled by turning the secure bit off and the checksum bit on. In a local area network that was considered to be both reliable and secure neither bit might be set.

Acknowledgement

I am grateful to Steve Kent for his little known technical note, "TCP and Communication Security," which provides an insightful exploration of the problems encountered in producing a secure version of TCP.

References

[1] D. L. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, Vol. 24, No. 2, pp. 84–88 , February1981.

[2] Whitfield Diffie, "Conventional Versus Public Key Cryptosystems," in *Secure Communiations and Asymetric Cryptosystems*, Edited by Gustavus J. Simmons, Westview Press, Boulder, Colorado, 1982.

[3] "DoD Standard, Internet Protocol," Information Sciences Institute, University of Southern California, Marina del Rey, California, RFC 791, September 1981.

[4] "DoD Standard, Transmission Control Protocol," Information Sciences Institute, University of Southern California, Marina del Rey, California, RFC 793, September 1981.

[5] Steven T. Kent, "Some Thoughts on TCP and Communication Security," MIT, Laboratory for Computer Science, Local Network Note, No. 6, 4 May1977.

[6] "Modes of Operation for the Data Encryption Standard," National Bureau of Standards, Federal Information Processing Standards Publication 81, 1980.