Internet Engineering Task Force (IETF)
Request for Comments: 5925
Obsoletes: 2385
Category: Standards Track
ISSN: 2070-1721

J. Touch
USC/ISI
A. Mankin
Johns Hopkins Univ.
R. Bonica
Juniper Networks

June 2010

The TCP Authentication Option

Abstract

This document specifies the TCP Authentication Option (TCP-AO), which obsoletes the TCP MD5 Signature option of RFC 2385 (TCP MD5). TCP-AO specifies the use of stronger Message Authentication Codes (MACs), protects against replays even for long-lived TCP connections, and provides more details on the association of security with TCP connections than TCP MD5. TCP-AO is compatible with either a static Master Key Tuple (MKT) configuration or an external, out-of-band MKT management mechanism; in either case, TCP-AO also protects connections when using the same MKT across repeated instances of a connection, using traffic keys derived from the MKT, and coordinates MKT changes between endpoints. The result is intended to support current infrastructure uses of TCP MD5, such as to protect long-lived connections (as used, e.g., in BGP and LDP), and to support a larger set of MACs with minimal other system and operational changes. TCP-AO uses a different option identifier than TCP MD5, even though TCP-AO and TCP MD5 are never permitted to be used simultaneously. TCP-AO supports IPv6, and is fully compatible with the proposed requirements for the replacement of TCP MD5.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5925.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The TCP MD5 Signature (TCP MD5) is a TCP option that authenticates TCP segments, including the TCP IPv4 pseudoheader, TCP header, and TCP data. It was developed to protect BGP sessions from spoofed TCP segments, which could affect BGP data or the robustness of the TCP connection itself [RFC2385][RFC4953].

There have been many recent concerns about TCP MD5. Its use of a simple keyed hash for authentication is problematic because there have been escalating attacks on the algorithm itself [Wa05]. TCP MD5 also lacks both key-management and algorithm agility. This document adds the latter, and provides a simple key coordination mechanism giving the ability to move from one key to another within the same connection. It does not however provide for complete cryptographic key management to be handled in band of TCP, because TCP SYN segments lack sufficient remaining space to handle such a negotiation (see Section 7.6). This document obsoletes the TCP MD5 option with a more general TCP Authentication Option (TCP-AO). This new option supports the use of other, stronger hash functions, provides replay protection for long-lived connections and across repeated instances of a single connection, coordinates key changes between endpoints, and provides a more explicit recommendation for external key management. The result is compatible with IPv6, and is fully compatible with proposed requirements for a replacement for TCP MD5 [Ed07].

TCP-AO obsoletes TCP MD5, although a particular implementation may support both mechanisms for backward compatibility. For a given connection, only one can be in use. TCP MD5-protected connections cannot be migrated to TCP-AO because TCP MD5 does not support any changes to a connection's security algorithm once established.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lowercase uses of these words are not to be interpreted as carrying RFC 2119 significance.

In this document, the characters ">>" preceeding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

1.2. Applicability Statement

TCP-AO is intended to support current uses of TCP MD5, such as to protect long-lived connections for routing protocols, such as BGP and LDP. It is also intended to provide similar protection to any long-lived TCP connection, as might be used between proxy caches, for example, and is not designed solely or primarily for routing protocol uses.

TCP-AO is intended to replace (and thus obsolete) the use of TCP MD5. TCP-AO enhances the capabilities of TCP MD5 as summarized in Section 1.3. This document recommends overall that:

>> TCP implementations that support TCP MD5 MUST support TCP-AO.

>> TCP-AO SHOULD be implemented where the protection afforded by TCP authentication is needed, because either IPsec is not supported or TCP-AO's particular properties are needed (e.g., per-connection keys).

>> TCP-AO MAY be implemented elsewhere.

TCP-AO is not intended to replace the use of the IPsec suite (IPsec and Internet Key Exchange Protocol (IKE)) to protect TCP connections [RFC4301][RFC4306]. Specific differences are noted in Section 1.3. In fact, we recommend the use of IPsec and IKE, especially where IKE's level of existing support for parameter negotiation, session key negotiation, or rekeying are desired. TCP-AO is intended for use only where the IPsec suite would not be feasible, e.g., as has been suggested is the case to support some routing protocols [RFC4953], or in cases where keys need to be tightly coordinated with individual transport sessions [Ed07].

TCP-AO is not intended to replace the use of Transport Layer Security (TLS) [RFC5246], Secure BGP (sBGP) or Secure Origin BGP (soBGP) [Le09], or any other mechanisms that protect only the TCP data stream. TCP-AO protects the transport layer, preventing attacks from disabling the TCP connection itself [RFC4953]. Data stream mechanisms protect only the contents of the TCP segments, and can be disrupted when the connection is affected. Some of these data protection protocols -- notably TLS -- offer a richer set of key management and authentication mechanisms than TCP-AO, and thus protect the data stream in a different way. TCP-AO may be used together with these data stream protections to complement each other's strengths.

1.3. Executive Summary

This document replaces TCP MD5 as follows [RFC2385]:

- TCP-AO uses a separate option Kind (29).
- TCP-AO allows TCP MD5 to continue to be used concurrently for legacy connections.
- TCP-AO replaces TCP MD5's single MAC algorithm with MACs specified in a separate document and can be extended to include other MACs.
- TCP-AO allows rekeying during a TCP connection, assuming that an out-of-band protocol or manual mechanism provides the new keys. The option includes a 'key ID', which allows the efficient concurrent use of multiple keys, and a key coordination mechanism using a 'receive next key ID' manages the key change within a connection. Note that TCP MD5 does not preclude rekeying during a connection, but does not require its support either. Further, TCP-AO supports key changes with zero segment loss, whereas key changes in TCP MD5 can lose segments in transit during the changeover or require trying multiple keys on each received segment during key use overlap because it lacks an explicit key ID. Although TCP recovers lost segments through retransmission, loss can have a substantial impact on performance.
- TCP-AO provides automatic replay protection for long-lived connections using sequence number extensions.
- TCP-AO ensures per-connection traffic keys as unique as the TCP connection itself, using TCP's Initial Sequence Numbers (ISNs) for differentiation, even when static master key tuples are used across repeated instances of connections on a single socket pair.
- TCP-AO specifies the details of how this option interacts with TCP's states, event processing, and user interface.
- TCP-AO is 2 bytes shorter than TCP MD5 (16 bytes overall, rather than 18) in the initially specified default case (using a 96-bit MAC).

TCP-AO differs from an IPsec/IKE solution as follows [RFC4301][RFC4306]:

- TCP-AO does not support dynamic parameter negotiation.

- TCP-AO includes TCP's socket pair (source address, destination address, source port, destination port) as a security parameter index (together with the KeyID), rather than using a separate field as an index (IPsec's Security Parameter Index (SPI)).

- TCP-AO forces a change of computed MACs when a connection restarts, even when reusing a TCP socket pair (IP addresses and port numbers) [Ed07].

- TCP-AO does not support encryption.

- TCP-AO does not authenticate ICMP messages (some ICMP messages may be authenticated when using IPsec, depending on the configuration).

2. The TCP Authentication Option

The TCP Authentication Option (TCP-AO) uses a TCP option Kind value of 29. The following sections describe TCP-AO and provide a review of TCP MD5 for comparison.

2.1. Review of TCP MD5 Option

For review, the TCP MD5 option is shown in Figure 1.

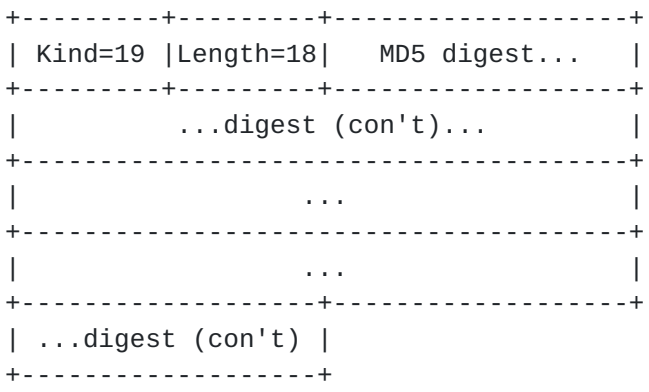

Figure 1: The TCP MD5 Option [RFC2385]

In the TCP MD5 option, the length is fixed, and the MD5 digest occupies 16 bytes following the Kind and Length fields (each one byte), using the full MD5 digest of 128 bits [RFC1321].

The TCP MD5 option specifies the use of the MD5 digest calculation over the following values in the following order:

1. The IP pseudoheader (IP source and destination addresses, protocol number, and segment length).

2. The TCP header excluding options and checksum.

3. The TCP data payload.

4. A key.

2.2. The TCP Authentication Option Format

TCP-AO provides a superset of the capabilities of TCP MD5, and is minimal in the spirit of SP4 [SDNS88]. TCP-AO uses a new Kind field, and similar Length field to TCP MD5, a KeyID field, and a RNextKeyID field as shown in Figure 2.

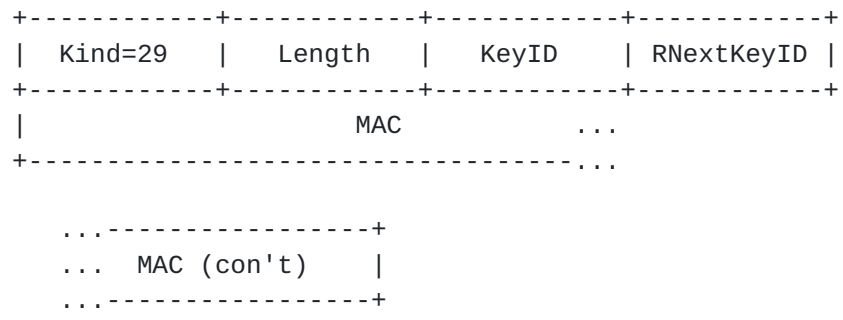

Figure 2: The TCP Authentication Option (TCP-AO)

TCP-AO defines these fields as follows:

o Kind: An unsigned 1-byte field indicating TCP-AO. TCP-AO uses a new Kind value of 29.

  >> An endpoint MUST NOT use TCP-AO for the same connection in which TCP MD5 is used. When both options appear, TCP MUST silently discard the segment.

  >> A single TCP segment MUST NOT have more than one TCP-AO in its options sequence. When multiple TCP-AOs appear, TCP MUST discard the segment.

o Length: An unsigned 1-byte field indicating the length of the option in bytes, including the Kind, Length, KeyID, RNextKeyID, and MAC fields.

  >> The Length value MUST be greater than or equal to 4. When the Length value is less than 4, TCP MUST discard the segment.

  >> The Length value MUST be consistent with the TCP header length. When the Length value is invalid, TCP MUST discard the segment.

This Length check implies that the sum of the sizes of all options, when added to the size of the base TCP header (5 words), matches the TCP Offset field exactly. This full verification can be computed because RFC 793 specifies the size of the required options, and RFC 1122 requires that all new options follow a common format with a fixed-length field location [RFC793][RFC1122]. A partial verification can be limited to check only TCP-AO, so that the TCP-AO length, when added to the TCP-AO offset from the start of the TCP header, does not exceed the TCP header size as indicated in the TCP header Offset field.

Values of 4 and other small values larger than 4 (e.g., indicating MAC fields of very short length) are of dubious utility but are not specifically prohibited.

- KeyID: An unsigned 1-byte field indicating the Master Key Tuple (MKT, as defined in Section 3.1) used to generate the traffic keys that were used to generate the MAC that authenticates this segment.

  It supports efficient key changes during a connection and/or to help with key coordination during connection establishment, to be discussed further in Section 6.1. Note that the KeyID has no cryptographic properties -- it need not be random, nor are there any reserved values.

  >> KeyID values MAY be the same in both directions of a connection, but do not have to be and there is no special meaning when they are.

  This allows MKTs to be installed on a set of devices without coordinating the KeyIDs across that entire set in advance, and allows new devices to be added to that set using a group of MKTs later without requiring renumbering of KeyIDs. These two capabilities are particularly important when used with wildcards in the TCP socket pair of the MKT, i.e., when an MKT is used among a set of devices specified by a pattern (as noted in Section 3.1).

- RNextKeyID: An unsigned 1-byte field indicating the MKT that is ready at the sender to be used to authenticate received segments, i.e., the desired 'receive next' key ID.

  It supports efficient key change coordination, to be discussed further in Section 6.1. Note that the RNextKeyID has no cryptographic properties -- it need not be random, nor are there any reserved values.

- MAC: Message Authentication Code. Its contents are determined by the particulars of the security association. Typical MACs are 96-128 bits (12-16 bytes), but any length that fits in the header of the segment being authenticated is allowed. The MAC computation is described further in Section 5.1.

  >> Required support for TCP-AO MACs is defined in [RFC5926]; other MACs MAY be supported.

TCP-AO fields do not indicate the MAC algorithm either implicitly (as with TCP MD5) or explicitly. The particular algorithm used is considered part of the configuration state of the connection's security and is managed separately (see Section 3).

Please note that the use of TCP-AO does not affect TCP's advertised Maximum Segment Size (MSS), as is the case for all TCP options [Bo09].

The remainder of this document explains how TCP-AO is handled and its relationship to TCP.

3. TCP-AO Keys and Their Properties

TCP-AO relies on two sets of keys to authenticate incoming and outgoing segments: Master Key Tuples (MKTs) and traffic keys. MKTs are used to derive unique traffic keys, and include the keying material used to generate those traffic keys, as well as indicating the associated parameters under which traffic keys are used. Such parameters include whether TCP options are authenticated, and indicators of the algorithms used for traffic key derivation and MAC calculation. Traffic keys are the keying material used to compute the MAC of individual TCP segments.

3.1. Master Key Tuple

A Master Key Tuple (MKT) describes TCP-AO properties to be associated with one or more connections. It is composed of the following:

- TCP connection identifier. A TCP socket pair, i.e., a local IP address, a remote IP address, a TCP local port, and a TCP remote port. Values can be partially specified using ranges (e.g., 2-30), masks (e.g., 0xF0), wildcards (e.g., "*"), or any other suitable indication.

- TCP option flag. This flag indicates whether TCP options other than TCP-AO are included in the MAC calculation. When options are included, the content of all options, in the order present, is included in the MAC, with TCP-AO's MAC field zeroed out. When the

options are not included, all options other than TCP-AO are excluded from all MAC calculations (skipped over, not zeroed). Note that TCP-AO, with its MAC field zeroed out, is always included in the MAC calculation, regardless of the setting of this flag; this protects the indication of the MAC length as well as the key ID fields (KeyID, RNextKeyID). The option flag applies to TCP options in both directions (incoming and outgoing segments).

- IDs. The values used in the KeyID or RNextKeyID of TCP-AO; used to differentiate MKTs in concurrent use (KeyID), as well as to indicate when MKTs are ready for use in the opposite direction (RNextKeyID).

  Each MKT has two IDs - -- a SendID and a RecvID. The SendID is inserted as the KeyID of the TCP-AO option of outgoing segments, and the RecvID is matched against the TCP-AO KeyID of incoming segments. These and other uses of these two IDs are described further in Sections 7.4 and 7.5.

  >> MKT IDs MUST support any value, 0-255 inclusive. There are no reserved ID values.

  ID values are assigned arbitrarily, i.e., the values are not monotonically increasing, have no reserved values, and are otherwise not meaningful. They can be assigned in sequence, or based on any method mutually agreed by the connection endpoints (e.g., using an external MKT management mechanism).

  >> IDs MUST NOT be assumed to be randomly assigned.

- Master key. A byte sequence used for generating traffic keys, this may be derived from a separate shared key by an external protocol over a separate channel. This sequence is used in the traffic key generation algorithm described in Section 5.2.

  Implementations are advised to keep master key values in a private, protected area of memory or other storage.

- Key Derivation Function (KDF). Indicates the key derivation function and its parameters, as used to generate traffic keys from master keys. It is explained further in Section 5.2 of this document and specified in detail in [RFC5926].

- Message Authentication Code (MAC) algorithm. Indicates the MAC algorithm and its parameters as used for this connection. It is explained further in Section 5.1 of this document and specified in detail in [RFC5926].

>> Components of an MKT MUST NOT change during a connection.

MKT component values cannot change during a connection because TCP state is coordinated during connection establishment. TCP lacks a handshake for modifying that state after a connection has been established.

>> The set of MKTs MAY change during a connection.

MKT parameters are not changed. Instead, new MKTs can be installed, and a connection can change which MKT it uses.

>> The IDs of MKTs MUST NOT overlap where their TCP connection identifiers overlap.

This document does not address how MKTs are created by users or processes. It is presumed that an MKT affecting a particular connection cannot be destroyed during an active connection -- or, equivalently, that its parameters are copied to an area local to the connection (i.e., instantiated) and so changes would affect only new connections. The MKTs can be managed by a separate application protocol.

3.2. Traffic Keys

A traffic key is a key derived from the MKT and the local and remote IP address pairs and TCP port numbers, and, for established connections, the TCP Initial Sequence Numbers (ISNs) in each direction. Segments exchanged before a connection is established use the same information, substituting zero for unknown values (e.g., ISNs not yet coordinated).

A single MKT can be used to derive any of four different traffic keys:

- Send_SYN_traffic_key
- Receive_SYN_traffic_key
- Send_other_traffic_key
- Receive_other_traffic_key

Note that the keys are unidirectional. A given connection typically uses only three of these keys, because only one of the SYN keys is typically used. All four are used only when a connection goes through 'simultaneous open' [RFC793].

The relationship between MKTs and traffic keys is shown in Figure 3. Traffic keys are indicated with a "*". Note that every MKT can be used to derive any of the four traffic keys, but only the keys actually needed to handle the segments of a connection need to be computed. Section 5.2 provides further details on how traffic keys are derived.

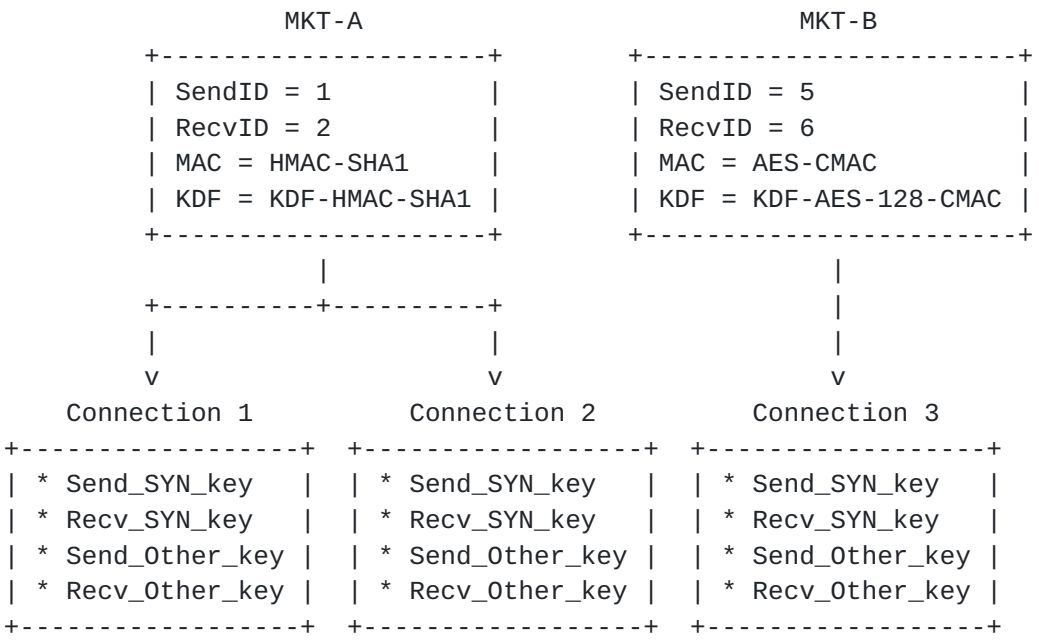

Figure 3: Relationship between MKTs and Traffic Keys

3.3. MKT Properties

TCP-AO requires that every protected TCP segment match exactly one MKT. When an outgoing segment matches an MKT, TCP-AO is used. When no match occurs, TCP-AO is not used. Multiple MKTs may match a single outgoing segment, e.g., when MKTs are being changed. Those MKTs cannot have conflicting IDs (as noted elsewhere), and some mechanism must determine which MKT to use for each given outgoing segment.

>> An outgoing TCP segment MUST match at most one desired MKT, indicated by the segment's socket pair. The segment MAY match multiple MKTs, provided that exactly one MKT is indicated as desired. Other information in the segment MAY be used to determine the desired MKT when multiple MKTs match; such information MUST NOT include values in any TCP option fields.

We recommend that the mechanism used to select from among multiple MKTs use only information that TCP-AO would authenticate. Because MKTs may indicate that options other than TCP-AO are ignored in the MAC calculation, we recommend that TCP options should not be used to determine MKTs.

>> An incoming TCP segment including TCP-AO MUST match exactly one MKT, indicated solely by the segment's socket pair and its TCP-AO KeyID.

Incoming segments include an indicator inside TCP-AO to select from among multiple matching MKTs -- the KeyID field. TCP-AO requires that the KeyID alone be used to differentiate multiple matching MKTs, so that MKT changes can be coordinated using the TCP-AO key change coordination mechanism.

>> When an outgoing TCP segment matches no MKTs, TCP-AO is not used.

TCP-AO is always used when outgoing segments match an MKT, and is not used otherwise.

4. Per-Connection TCP-AO Parameters

TCP-AO uses a small number of parameters associated with each connection that uses TCP-AO, once instantiated. These values can be stored in the Transport Control Block (TCB) [RFC793]. These values are explained in subsequent sections of this document as noted; they include:

1. Current_key - the MKT currently used to authenticate outgoing segments, whose SendID is inserted in outgoing segments as KeyID (see Section 7.4, step 2.f). Incoming segments are authenticated using the MKT corresponding to the segment and its TCP-AO KeyID (see Section 7.5, step 2.c), as matched against the MKT TCP connection identifier and the MKT RecvID. There is only one current_key at any given time on a particular connection.

   >> Every TCP connection in a non-IDLE state MUST have at most one current_key specified.

2. Rnext_key - the MKT currently preferred for incoming (received) segments, whose RecvID is inserted in outgoing segments as RNextKeyID (see Section 7.4, step 2.d).

   >> Each TCP connection in a non-IDLE state MUST have at most one rnext_key specified.

3. A pair of Sequence Number Extensions (SNEs). SNEs are used to prevent replay attacks, as described in Section 6.2. Each SNE is initialized to zero upon connection establishment. Its use in the MAC calculation is described in Section 5.1.

4. One or more MKTs. These are the MKTs that match this connection's socket pair.

MKTs are used, together with other parameters of a connection, to create traffic keys unique to each connection, as described in Section 5.2. These traffic keys can be cached after computation, and can be stored in the TCB with the corresponding MKT information. They can be considered part of the per-connection parameters.

5. Cryptographic Algorithms

TCP-AO uses cryptographic algorithms to compute the MAC (Message Authentication Code) that is used to authenticate a segment and its headers; these are called MAC algorithms and are specified in a separate document to facilitate updating the algorithm requirements independently from the protocol [RFC5926]. TCP-AO also uses cryptographic algorithms to convert MKTs, which can be shared across connections, into unique traffic keys for each connection. These are called Key Derivation Functions (KDFs) and are specified [RFC5926]. This section describes how these algorithms are used by TCP-AO.

5.1. MAC Algorithms

MAC algorithms take a variable-length input and a key and output a fixed-length number. This number is used to determine whether the input comes from a source with that same key, and whether the input has been tampered with in transit. MACs for TCP-AO have the following interface:

```
MAC = MAC_alg(traffic_key, message)

INPUT: MAC_alg, traffic_key, message

OUTPUT: MAC
```

where:

- MAC_alg - the specific MAC algorithm used for this computation. The MAC algorithm specifies the output length, so no separate output length parameter is required. This is specified as described in [RFC5926].

- Traffic_key - traffic key used for this computation. This is computed from the connection's current MKT as described in Section 5.2.

- Message - input data over which the MAC is computed. In TCP-AO, this is the TCP segment prepended by the IP pseudoheader and TCP header options, as described in Section 5.1.

- MAC - the fixed-length output of the MAC algorithm, given the parameters provided.

At the time of this writing, the algorithms' definitions for use in TCP-AO, as described in [RFC5926], are each truncated to 96 bits. Though the algorithms each output a larger MAC, 96 bits provides a reasonable trade-off between security and message size. However, this could change in the future, so TCP-AO size should not be assumed as fixed length.

The MAC algorithm employed for the MAC computation on a connection is done so by definition in the MKT, per the definition in [RFC5926].

The mandatory-to-implement MAC algorithms for use with TCP-AO are described in a separate RFC [RFC5926]. This allows the TCP-AO specification to proceed along the IETF Standards Track even if changes are needed to its associated algorithms and their labels (as might be used in a user interface or automated MKT management protocol) as a result of the ever evolving world of cryptography.

>> Additional algorithms, beyond those mandated for TCP-AO, MAY be supported.

The data input to the MAC is in the following fields in the following sequence, interpreted in network-standard byte order:

1. The Sequence Number Extension (SNE), in network-standard byte order, as follows (described further in Section 6.2):

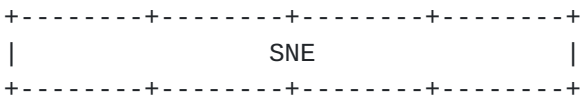

Figure 4: Sequence Number Extension

   The SNE for transmitted segments is maintained locally in the SND.SNE value; for received segments, a local RCV.SNE value is used. The details of how these values are maintained and used are in Sections 6.2, 7.4, and 7.5.

2. The IP pseudoheader: IP source and destination addresses, protocol number, and segment length, all in network byte order, prepended to the TCP header below. The IP pseudoheader is exactly as used for the TCP checksum in either IPv4 or IPv6 [RFC793][RFC2460]:

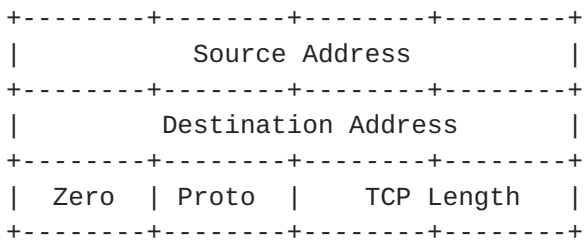

Figure 5: TCP IPv4 Pseudoheader [RFC793]

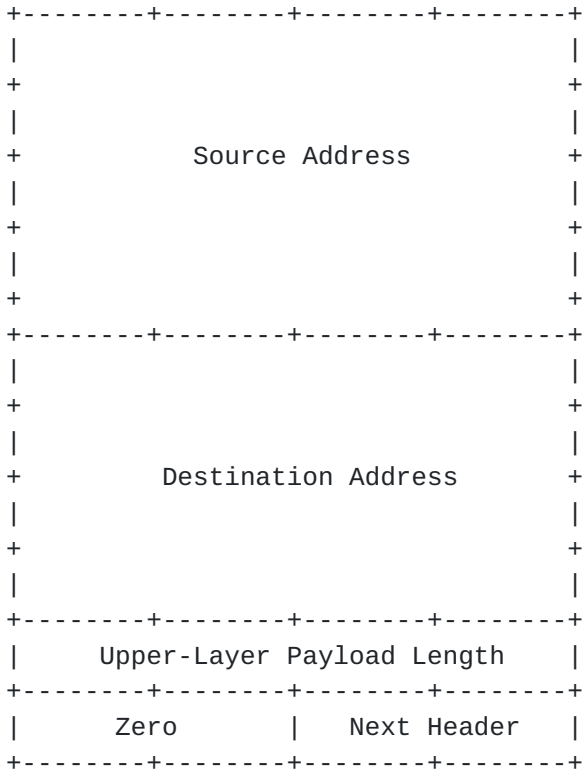

Figure 6: TCP IPv6 Pseudoheader [RFC2460]

3. The TCP header, by default including options, and where the TCP checksum and TCP-AO MAC fields are set to zero, all in network-byte order.

   The TCP option flag of the MKT indicates whether the TCP options are included in the MAC. When included, only the TCP-AO MAC field is zeroed.

When TCP options are not included, all TCP options except for TCP-AO are omitted from MAC processing. Again, the TCP-AO MAC field is zeroed for the MAC processing.

4. The TCP data, i.e., the payload of the TCP segment.

Note that the traffic key is not included as part of the data; the MAC algorithm indicates how to use the traffic key, for example, as HMACs do [RFC2104][RFC2403]. The traffic key is derived from the current MKT as described in Section 5.2.

5.2. Traffic Key Derivation Functions

TCP-AO's traffic keys are derived from the MKTs using Key Derivation Functions (KDFs). The KDFs used in TCP-AO have the following interface:

```
traffic_key = KDF_alg(master_key, context, output_length)

INPUT: KDF_alg, master_key, context, output_length

OUTPUT: traffic_key
```

where:

- KDF_alg - The specific Key Derivation Function (KDF) that is the basic building block used in constructing the traffic key, as indicated in the MKT. This is specified as described in [RFC5926].

- Master_key - The master_key string, as will be stored into the associated MKT.

- Context - The context used as input in constructing the traffic_key, as specified in [RFC5926]. The specific way this context is used, in conjunction with other information, to create the raw input to the KDF is also explained further in [RFC5926].

- Output_length - The desired output length of the KDF, i.e., the length to which the KDF's output will be truncated. This is specified as described in [RFC5926].

- Traffic_key - The desired output of the KDF, of length output_length, to be used as input to the MAC algorithm, as described in Section 5.1.

The context used as input to the KDF combines the TCP socket pair with the endpoint Initial Sequence Numbers (ISNs) of a connection. This data is unique to each TCP connection instance, which enables TCP-AO to generate unique traffic keys for that connection, even from an MKT used across many different connections or across repeated connections that share a socket pair. Unique traffic keys are generated without relying on external key management properties. The KDF context is defined in Figures 7 and 8.

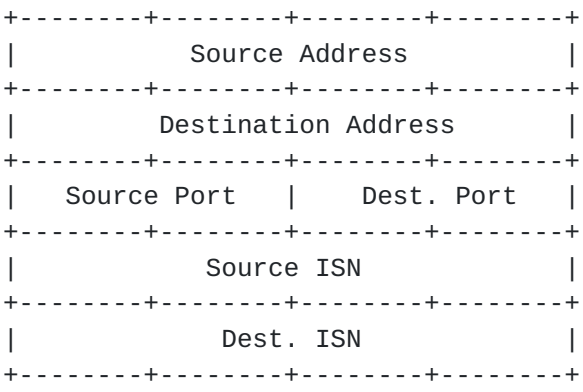

Figure 7: KDF Context for an IPv4 Connection

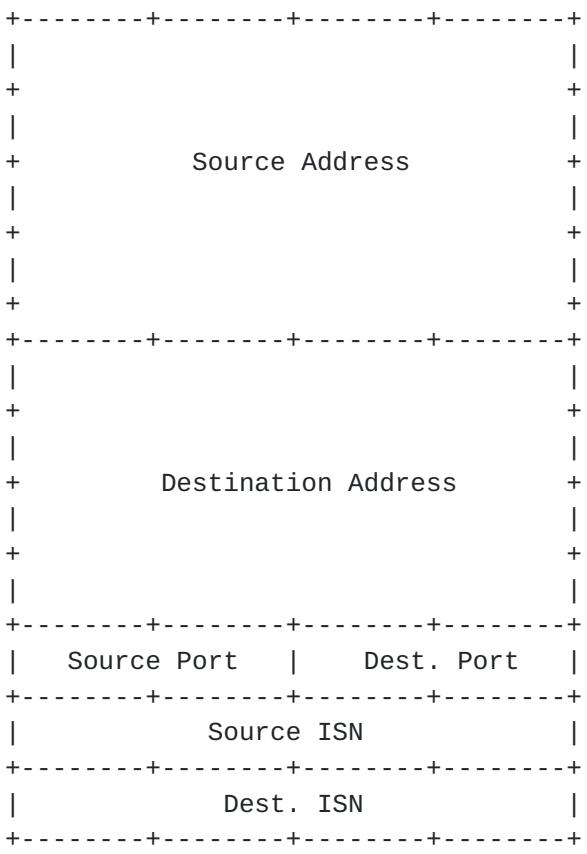

Figure 8: KDF Context for an IPv6 Connection

Traffic keys are directional, so "source" and "destination" are interpreted differently for incoming and outgoing segments. For incoming segments, source is the remote side; whereas for outgoing segments, source is the local side. This further ensures that connection keys generated for each direction are unique.

For SYN segments (segments with the SYN set, but the ACK not set), the destination ISN is not known. For these segments, the connection key is computed using the context shown above, in which the destination ISN value is zero. For all other segments, the ISN pair is used when known. If the ISN pair is not known, e.g., when sending a reset (RST) after a reboot, the segment should be sent without authentication; if authentication was required, the segment cannot have been MAC'd properly anyway and would have been dropped on receipt.

>> TCP-AO SYN segments (SYN set, no ACK set) MUST use a destination ISN of zero (whether sent or received); all other segments use the known ISN pair.

Overall, this means that each connection will use up to four distinct traffic keys for each MKT:

- Send_SYN_traffic_key - the traffic key used to authenticate outgoing SYNs. The source ISN is known (the TCP connection's local ISN), and the destination (remote) ISN is unknown (and so the value 0 is used).

- Receive_SYN_traffic_key - the traffic key used to authenticate incoming SYNs. The source ISN is known (the TCP connection's remote ISN), and the destination (remote) ISN is unknown (and so the value 0 is used).

- Send_other_traffic_key - the traffic key used to authenticate all other outgoing TCP segments.

- Receive_other_traffic_key - the traffic key used to authenticate all other incoming TCP segments.

The following table describes how each of these traffic keys is computed, where the TCP-AO algorithms refer to source (S) and destination (D) values of the IP address, TCP port, and ISN, and each segment (incoming or outgoing) has a value that refers to the local side of the connection (l) and remote side (r):

	S-IP	S-port	S-ISN	D-IP	D-port	D-ISN
Send_SYN_traffic_key	l-IP	l-port	l-ISN	r-IP	r-port	0
Receive_SYN_traffic_key	r-IP	r-port	r-ISN	l-IP	l-port	0
Send_other_traffic_key	l-IP	l-port	l-ISN	r-IP	r-port	r-ISN
Receive_other_traffic_key	r-IP	r-port	r-ISN	l-IP	l-port	l-ISN

The use of both ISNs in the traffic key computations ensures that segments cannot be replayed across repeated connections reusing the same socket; their 32-bit space avoids repeated use except under reboot, and reuse assumes both sides repeat their use on the same connection. We do expect that:

>> Endpoints should select ISNs pseudorandomly, e.g., as in [RFC1948].

A SYN is authenticated using a destination ISN of zero (whether sent or received), and all other segments would be authenticated using the ISN pair for the connection. There are other cases in which the destination ISN is not known, but segments are emitted, such as after an endpoint reboots, when it is possible that the two endpoints would not have enough information to authenticate segments. This is addressed further in Section 7.7.

5.3. Traffic Key Establishment and Duration Issues

TCP-AO does not provide a mechanism for traffic key negotiation or parameter negotiation (MAC algorithm, length, or use of TCP-AO on a connection), or for coordinating rekeying during a connection. We assume out-of-band mechanisms for MKT establishment, parameter negotiation, and rekeying. This separation of MKT use from MKT management is similar to that in the IPsec suite [RFC4301][RFC4306].

We encourage users of TCP-AO to apply known techniques for generating appropriate MKTs, including the use of reasonable master key lengths, limited traffic key sharing, and limiting the duration of MKT use [RFC3562]. This also includes the use of per-connection nonces, as suggested in Section 5.2.

TCP-AO supports rekeying in which new MKTs are negotiated and coordinated out of band, either via a protocol or a manual procedure [RFC4808]. New MKT use is coordinated using the out-of-band mechanism to update both TCP endpoints. When only a single MKT is used at a time, the temporary use of invalid MKTs could result in segments being dropped; although TCP is already robust to such drops, TCP-AO uses the KeyID field to avoid such drops. A given connection can have multiple matching MKTs, where the KeyID field is used to identify the MKT that corresponds to the traffic key used for a segment, to avoid the need for expensive trial-and-error testing of MKTs in sequence.

TCP-AO provides an explicit MKT coordination mechanism, described in Section 6.1. Such a mechanism is useful when new MKTs are installed, or when MKTs are changed, to determine when to commence using installed MKTs.

Users are advised to manage MKTs following the spirit of the advice for key management when using TCP MD5 [RFC3562], notably to use appropriate key lengths (12-24 bytes) and to avoid sharing MKTs among multiple BGP peering arrangements.

5.3.1. MKT Reuse Across Socket Pairs

MKTs can be reused across different socket pairs within a host, or across different instances of a socket pair within a host. In either case, replay protection is maintained.

MKTs reused across different socket pairs cannot enable replay attacks because the TCP socket pair is included in the MAC, as well as in the generation of the traffic key. MKTs reused across repeated

instances of a given socket pair cannot enable replay attacks because the connection ISNs are included in the traffic key generation algorithm, and ISN pairs are unlikely to repeat over useful periods.

5.3.2. MKTs Use within a Long-Lived Connection

TCP-AO uses Sequence Number Extensions (SNEs) to prevent replay attacks within long-lived connections. Explicit MKT rollover, accomplished by external means and indexed using the KeyID field, can be used to change keying material for various reasons (e.g., personnel turnover), but is not required to support long-lived connections.

6. Additional Security Mechanisms

TCP-AO adds mechanisms to support efficient use, especially in environments where only manual keying is available. These include the previously described mechanisms for supporting multiple concurrent MKTs (via the KeyID field) and for generating unique per-connection traffic keys (via the KDF). This section describes additional mechanisms to coordinate MKT changes and to prevent replay attacks when a traffic key is not changed for long periods of time.

6.1. Coordinating Use of New MKTs

At any given time, a single TCP connection may have multiple MKTs specified for each segment direction (incoming, outgoing). TCP-AO provides a mechanism to indicate when a new MKT is ready, which allows the sender to commence use of that new MKT. This mechanism allows new MKT use to be coordinated, to avoid unnecessary loss due to sender authentication using an MKT not yet ready at the receiver.

Note that this is intended as an optimization. Deciding when to start using a key is a performance issue. Deciding when to remove an MKT is a security issue. Invalid MKTs are expected to be removed. TCP-AO provides no mechanism to coordinate their removal, as we consider this a key management operation.

New MKT use is coordinated through two ID fields in the header:

- KeyID
- RNextKeyID

KeyID represents the outgoing MKT information used by the segment sender to create the segment's MAC (outgoing), and the corresponding incoming keying information used by the segment receiver to validate that MAC. It contains the SendID of the MKT in active use in that direction.

RNextKeyID represents the preferred MKT information to be used for subsequent received segments ('receive next'). That is, it is a way for the segment sender to indicate a ready incoming MKT for future segments it receives, so that the segment receiver can know when to switch MKTs (and thus their KeyIDs and associated traffic keys). It indicates the RecvID of the MKT desired for incoming segments.

There are two pointers kept by each side of a connection, as noted in the per-connection information (see Section 4):

o Currently active outgoing MKT (current_key)

o Current preference for incoming MKT (rnext_key)

Current_key indicates an MKT that is used to authenticate outgoing segments. Upon connection establishment, it points to the first MKT selected for use.

Rnext_key points to an incoming MKT that is ready and preferred for use. Upon connection establishment, this points to the currently active incoming MKT. It can be changed when new MKTs are installed (e.g., by either automatic MKT management protocol operation or user manual selection).

Rnext_key is changed only by manual user intervention or MKT management protocol operation. It is not manipulated by TCP-AO. Current_key is updated by TCP-AO when processing received TCP segments as discussed in the segment processing description in Section 7.5. Note that the algorithm allows the current_key to change to a new MKT, then change back to a previously used MKT (known as "backing up"). This can occur during an MKT change when segments are received out of order, and is considered a feature of TCP-AO, because reordering does not result in drops. The only way to avoid reuse of previously used MKTs is to remove the MKT when it is no longer considered permitted.

6.2. Preventing Replay Attacks within Long-Lived Connections

TCP uses a 32-bit sequence number, which may, for long-lived connections, roll over and repeat. This could result in TCP segments being intentionally and legitimately replayed within a connection. TCP-AO prevents replay attacks, and thus requires a way to

differentiate these legitimate replays from each other, and so it adds a 32-bit Sequence Number Extension (SNE) for transmitted and received segments.

The SNE extends the TCP sequence number so that segments within a single connection are always unique. When the TCP's sequence number rolls over, there is a chance that a segment could be repeated in total; using an SNE differentiates even identical segments sent with identical sequence numbers at different times in a connection. TCP-AO emulates a 64-bit sequence number space by inferring when to increment the high-order 32-bit portion (the SNE) based on transitions in the low-order portion (the TCP sequence number).

TCP-AO thus maintains SND.SNE for transmitted segments, and RCV.SNE for received segments, both initialized as zero when a connection begins. The intent of these SNEs is, together with TCP's 32-bit sequence numbers, to provide a 64-bit overall sequence number space.

For transmitted segments, SND.SNE can be implemented by extending TCP's sequence number to 64 bits; SND.SNE would be the top (high-order) 32 bits of that number. For received segments, TCP-AO needs to emulate the use of a 64-bit number space and correctly infer the appropriate high-order 32-bits of that number as RCV.SNE from the received 32-bit sequence number and the current connection context.

The implementation of SNEs is not specified in this document, but one possible way is described here that can be used for either RCV.SNE, SND.SNE, or both.

Consider an implementation with two SNEs as required (SND.SNE, RCV.SNE), and additional variables as listed below, all initialized to zero, as well as a current TCP segment field (SEG.SEQ):

- SND.PREV_SEQ, needed to detect rollover of SND.SEQ
- RCV.PREV_SEQ, needed to detect rollover of RCV.SEQ
- SND.SNE_FLAG, which indicates when to increment the SND.SNE
- RCV.SNE_FLAG, which indicates when to increment the RCV.SNE

When a segment is received, the following algorithm (in C-like pseudocode) computes the SNE used in the MAC; this is the "RCV" side, and an equivalent algorithm can be applied to the "SND" side:

```
/* set the flag when the SEG.SEQ first rolls over */
if ((RCV.SNE_FLAG == 0)
   && (RCV.PREV_SEQ > 0x7fff) && (SEG.SEQ < 0x7fff)) {
      RCV.SNE = RCV.SNE + 1;
      RCV.SNE_FLAG = 1;
}
/* decide which SNE to use after incremented */
if ((RCV.SNE_FLAG == 1) && (SEG.SEQ > 0x7fff)) {
   SNE = RCV.SNE - 1; # use the pre-increment value
} else {
   SNE = RCV.SNE; # use the current value
}
/* reset the flag in the *middle* of the window */
if ((RCV.PREV_SEQ < 0x7fff) && (SEG.SEQ > 0x7fff)) {
   RCV.SNE_FLAG = 0;
}
/* save the current SEQ for the next time through the code */
RCV.PREV_SEQ = SEG.SEQ;
```

In the above code, the first time the sequence number rolls over, i.e., when the new number is low (in the bottom half of the number space) and the old number is high (in the top half of the number space), the SNE is incremented and a flag is set.

If the flag is set and a high number is seen, it must be a reordered segment, so use the pre-increment SNE; otherwise, use the current SNE.

The flag will be cleared by the time the number rolls all the way around.

The flag prevents the SNE from being incremented again until the flag is reset, which happens in the middle of the window (when the old number is in the bottom half and the new is in the top half). Because the receive window is never larger than half of the number space, it is impossible to both set and reset the flag at the same time -- outstanding segments, regardless of reordering, cannot straddle both regions simultaneously.

7. TCP-AO Interaction with TCP

The following is a description of how TCP-AO affects various TCP states, segments, events, and interfaces. This description is intended to augment the description of TCP as provided in RFC 793, and its presentation mirrors that of RFC 793 as a result [RFC793].

7.1. TCP User Interface

The TCP user interface supports active and passive OPEN, SEND, RECEIVE, CLOSE, STATUS, and ABORT commands. TCP-AO does not alter this interface as it applies to TCP, but some commands or command sequences of the interface need to be modified to support TCP-AO. TCP-AO does not specify the details of how this is achieved.

TCP-AO requires that the TCP user interface be extended to allow the MKTs to be configured, as well as to allow an ongoing connection to manage which MKTs are active. The MKTs need to be configured prior to connection establishment, and the set of MKTs may change during a connection:

>> TCP OPEN, or the sequence of commands that configure a connection to be in the active or passive OPEN state, MUST be augmented so that an MKT can be configured.

>> A TCP-AO implementation MUST allow the set of MKTs for ongoing TCP connections (i.e., not in the CLOSED state) to be modified.

The MKTs associated with a connection need to be available for confirmation; this includes the ability to read the MKTs:

>> TCP STATUS SHOULD be augmented to allow the MKTs of a current or pending connection to be read (for confirmation).

Senders may need to be able to determine when the outgoing MKT changes (KeyID) or when a new preferred MKT (RNextKeyID) is indicated; these changes immediately affect all subsequent outgoing segments:

>> TCP SEND, or a sequence of commands resulting in a SEND, MUST be augmented so that the preferred outgoing MKT (current_key) and/or the preferred incoming MKT (rnext_key) of a connection can be indicated.

It may be useful to change the outgoing active MKT (current_key) even when no data is being sent, which can be achieved by sending a zero-length buffer or by using a non-send interface (e.g., socket options in Unix), depending on the implementation.

It is also useful to indicate recent segment KeyID and RNextKeyID values received; although there could be a number of such values, they are not expected to change quickly, so any recent sample should be sufficient:

>> TCP RECEIVE, or the sequence of commands resulting in a RECEIVE, MUST be augmented so that the KeyID and RNextKeyID of a recently received segment is available to the user out of band (e.g., as an additional parameter to RECEIVE or via a STATUS call).

7.2. TCP States and Transitions

TCP includes the states LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and CLOSED.

>> An MKT MAY be associated with any TCP state.

7.3. TCP Segments

TCP includes control (at least one of SYN, FIN, RST flags set) and data (none of SYN, FIN, or RST flags set) segments. Note that some control segments can include data (e.g., SYN).

>> All TCP segments MUST be checked against the set of MKTs for matching TCP connection identifiers.

>> TCP segments whose TCP-AO does not validate MUST be silently discarded.

>> A TCP-AO implementation MUST allow for configuration of the behavior of segments with TCP-AO but that do not match an MKT. The initial default of this configuration SHOULD be to silently accept such connections. If this is not the desired case, an MKT can be included to match such connections, or the connection can indicate that TCP-AO is required. Alternately, the configuration can be changed to discard segments with the AO option not matching an MKT.

>> Silent discard events SHOULD be signaled to the user as a warning, and silent accept events MAY be signaled to the user as a warning. Both warnings, if available, MUST be accessible via the STATUS interface. Either signal MAY be asynchronous, but if so, they MUST be rate-limited. Either signal MAY be logged; logging SHOULD allow rate-limiting as well.

All TCP-AO processing occurs between the interface of TCP and IP; for incoming segments, this occurs after validation of the TCP checksum. For outgoing segments, this occurs before computation of the TCP checksum.

Note that use of TCP-AO on a connection is not negotiated within TCP. It is the responsibility of the receiver to determine when TCP-AO is required via other means (e.g., out of band, manually or with a key management protocol) and to enforce that requirement.

7.4. Sending TCP Segments

The following procedure describes the modifications to TCP to support inserting TCP-AO when a segment departs.

>> Note that TCP-AO MUST be the last TCP option processed on outgoing segments, because its MAC calculation may include the values of other TCP options.

1. Find the per-connection parameters for the segment:

   a. If the segment is a SYN, then this is the first segment of a new connection. Find the matching MKT for this segment based on the segment's socket pair.

      i. If there is no matching MKT, omit TCP-AO. Proceed with transmitting the segment.

      ii. If there is a matching MKT, then set the per-connection parameters as needed (see Section 4). Proceed with the step 2.

   b. If the segment is not a SYN, then determine whether TCP-AO is being used for the connection and use the MKT as indicated by the current_key value from the per-connection parameters (see Section 4) and proceed with the step 2.

2. Using the per-connection parameters:

   a. Augment the TCP header with TCP-AO, inserting the appropriate Length and KeyID based on the MKT indicated by current_key (using the current_key MKT's SendID as the TCP-AO KeyID). Update the TCP header length accordingly.

   b. Determine SND.SNE as described in Section 6.2.

   c. Determine the appropriate traffic key, i.e., as pointed to by the current_key (as noted in Section 6.1, and as probably cached in the TCB). That is, use the send_SYN_traffic_key for SYN segments and the send_other_traffic_key for other segments.

d. Determine the RNextKeyID as indicated by the rnext_key pointer, and insert it in the TCP-AO RNextKeyID field (using the rnext_key MKT's RecvID as the TCP-AO KeyID) (as noted in Section 6.1).

e. Compute the MAC using the MKT (and cached traffic key) and data from the segment as specified in Section 5.1.

f. Insert the MAC in the TCP-AO MAC field.

g. Proceed with transmitting the segment.

7.5. Receiving TCP Segments

The following procedure describes the modifications to TCP to support TCP-AO when a segment arrives.

>> Note that TCP-AO MUST be the first TCP option processed on incoming segments, because its MAC calculation may include the values of other TCP options that could change during TCP option processing. This also protects the behavior of all other TCP options from the impact of spoofed segments or modified header information.

>> Note that TCP-AO checks MUST be performed for all incoming SYNs to avoid accepting SYNs lacking TCP-AO where required. Other segments can cache whether TCP-AO is needed in the TCB.

1. Find the per-connection parameters for the segment:

   a. If the segment is a SYN, then this is the first segment of a new connection. Find the matching MKT for this segment, using the segment's socket pair and its TCP-AO KeyID, matched against the MKT's TCP connection identifier and the MKT's RecvID.

      i. If there is no matching MKT, remove TCP-AO from the segment. Proceed with further TCP handling of the segment.

         NOTE: this presumes that connections that do not match any MKT should be silently accepted, as noted in Section 7.3.

      ii. If there is a matching MKT, then set the per-connection parameters as needed (see Section 4). Proceed with step 2.

2. Using the per-connection parameters:

   a. Check that the segment's TCP-AO Length matches the length indicated by the MKT.

      i. If the lengths differ, silently discard the segment. Log and/or signal the event as indicated in Section 7.3.

   b. Determine the segment's RCV.SNE as described in Section 6.2.

   c. Determine the segment's traffic key from the MKT as described in Section 5.1 (and as likely cached in the TCB). That is, use the receive_SYN_traffic_key for SYN segments and the receive_other_traffic_key for other segments.

   d. Compute the segment's MAC using the MKT (and its derived traffic key) and portions of the segment as indicated in Section 5.1.

      i. If the computed MAC differs from the TCP-AO MAC field value, silently discard the segment. Log and/or signal the event as indicated in Section 7.3.

   e. Compare the received RNextKeyID value to the currently active outgoing KeyID value (current_key MKT's SendID).

      i. If they match, no further action is required.

      ii. If they differ, determine whether the RNextKeyID MKT is ready.

         1. If the MKT corresponding to the segment's socket pair and RNextKeyID is not available, no action is required (RNextKeyID of a received segment needs to match the MKT's SendID).

         2. If the matching MKT corresponding to the segment's socket pair and RNextKeyID is available:

            a. Set current_key to the RNextKeyID MKT.

   f. Proceed with TCP processing of the segment.

It is suggested that TCP-AO implementations validate a segment's Length field before computing a MAC to reduce the overhead incurred by spoofed segments with invalid TCP-AO fields.

Additional reductions in MAC validation overhead can be supported in the MAC algorithms, e.g., by using a computation algorithm that prepends a fixed value to the computed portion and a corresponding validation algorithm that verifies the fixed value before investing in the computed portion. Such optimizations would be contained in the MAC algorithm specification, and thus are not specified in TCP-AO explicitly. Note that the KeyID cannot be used for connection validation per se, because it is not assumed random.

7.6. Impact on TCP Header Size

TCP-AO, using the initially required 96-bit MACs, uses a total of 16 bytes of TCP header space [RFC5926]. TCP-AO is thus 2 bytes smaller than the TCP MD5 option (18 bytes).

Note that the TCP option space is most critical in SYN segments, because flags in those segments could potentially increase the option space area in other segments. Because TCP ignores unknown segments, however, it is not possible to extend the option space of SYNs without breaking backward compatibility.

TCP's 4-bit data offset requires that the options end 60 bytes (15 32-bit words) after the header begins, including the 20-byte header. This leaves 40 bytes for options, of which 15 are expected in current implementations (listed below), leaving at most 25 for other uses. TCP-AO consumes 16 bytes, leaving 9 bytes for additional SYN options (depending on implementation dependant alignment padding, which could consume another 2 bytes at most).

- SACK permitted (2 bytes) [RFC2018][RFC3517]

- Timestamps (10 bytes) [RFC1323]

- Window scale (3 bytes) [RFC1323]

After a SYN, the following options are expected in current implementations of TCP:

- SACK (10bytes) [RFC2018][RFC3517] (18 bytes if D-SACK [RFC2883])

- Timestamps (10 bytes) [RFC1323]

TCP-AO continues to consume 16 bytes in non-SYN segments, leaving a total of 24 bytes for other options, of which the timestamp consumes 10. This leaves 14 bytes, of which 10 are used for a single SACK block. When two SACK blocks are used, such as to handle D-SACK, a smaller TCP-AO MAC would be required to make room for the additional SACK block (i.e., to leave 18 bytes for the D-SACK variant of the

SACK option) [RFC2883]. Note that D-SACK is not supportable in TCP MD5 in the presence of timestamps, because TCP MD5's MAC length is fixed and too large to leave sufficient option space.

Although TCP option space is limited, we believe TCP-AO is consistent with the desire to authenticate TCP at the connection level for similar uses as were intended by TCP MD5.

7.7. Connectionless Resets

TCP-AO allows TCP resets (RSTs) to be exchanged provided both sides have established valid connection state. After such state is established, if one side reboots, TCP-AO prevents TCP's RST mechanism from clearing out old state on the side that did not reboot. This happens because the rebooting side has lost its connection state, and thus its traffic keys.

It is important that implementations are capable of detecting excesses of TCP connections in such a configuration and can clear them out if needed to protect its memory usage [Ba10]. To protect against such state from accumulating and not being cleared out, a number of recommendations are made:

>> Connections using TCP-AO SHOULD also use TCP keepalives [RFC1122].

The use of TCP keepalives ensures that connections whose keys are lost are terminated after a finite time; a similar effect can be achieved at the application layer, e.g., with BGP keepalives [RFC4271]. Either kind of keepalive helps ensure the TCP state is cleared out in such a case; the alternative, of allowing unauthenticated RSTs to be received, would allow one of the primary vulnerabilities that TCP-AO is intended to prevent.

Keepalives ensure that connections are dropped across reboots, but this can have a detrimental effect on some protocols. Specifically, BGP reacts poorly to such connection drops, even if caused by the use of BGP keepalives; "graceful restart" was introduced to address this effect [RFC4724], and extended to support BGP with MPLS [RFC4781]. As a result:

>> BGP connections SHOULD require support for graceful restart when using TCP-AO.

We recognize that support for graceful restart is not always feasible. As a result:

>> When BGP without graceful restart is used with TCP-AO, both sides of the connection SHOULD save traffic keys in storage that persists across reboots and restore them after a reboot, and SHOULD limit any performance impacts that result from this storage/restoration.

7.8. ICMP Handling

TCP can be attacked both in band, using TCP segments, or out of band using ICMP. ICMP packets cannot be protected using TCP-AO mechanisms; however, in this way, both TCP-AO and IPsec do not directly solve the need for protected ICMP signaling. TCP-AO does make specific recommendations on how to handle certain ICMPs, beyond what IPsec requires, and these are made possible because TCP-AO operates inside the context of a TCP connection.

IPsec makes recommendations regarding dropping ICMPs in certain contexts or requiring that they are endpoint authenticated in others [RFC4301]. There are other mechanisms proposed to reduce the impact of ICMP attacks by further validating ICMP contents and changing the effect of some messages based on TCP state, but these do not provide the level of authentication for ICMP that TCP-AO provides for TCP [Go10]. As a result, we recommend a conservative approach to accepting ICMP messages as summarized in [Go10]:

>> A TCP-AO implementation MUST default to ignore incoming ICMPv4 messages of Type 3 (destination unreachable), Codes 2-4 (protocol unreachable, port unreachable, and fragmentation needed -- 'hard errors'), and ICMPv6 Type 1 (destination unreachable), Code 1 (administratively prohibited) and Code 4 (port unreachable) intended for connections in synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT) that match MKTs.

>> A TCP-AO implementation SHOULD allow whether such ICMPs are ignored to be configured on a per-connection basis.

>> A TCP-AO implementation SHOULD implement measures to protect ICMP "packet too big" messages, some examples of which are discussed in [Go10].

>> An implementation SHOULD allow ignored ICMPs to be logged.

This control affects only ICMPs that currently require 'hard errors', which would abort the TCP connection [RFC1122]. This recommendation is intended to be similar to how IPsec would handle those messages, with an additional default assumed [RFC4301].

8. Obsoleting TCP MD5 and Legacy Interactions

TCP-AO obsoletes TCP MD5. As we have noted earlier:

>> TCP implementations that support TCP MD5 MUST support TCP-AO.

Systems implementing TCP MD5 only are considered legacy, and ought to be upgraded when possible. In order to support interoperation with such legacy systems until upgrades are available:

>> TCP MD5 SHOULD be supported where interactions with legacy systems are needed.

>> A system that supports both TCP-AO and TCP MD5 MUST use TCP-AO for connections unless not supported by its peer, at which point it MAY use TCP MD5 instead.

>> A TCP implementation MUST NOT use both TCP-AO and TCP MD5 for a particular TCP connection, but MAY support TCP-AO and TCP MD5 simultaneously for different connections (notably to support legacy use of TCP MD5).

The Kind value explicitly indicates whether TCP-AO or TCP MD5 is used for a particular connection in TCP segments.

It is possible that MKTs could be augmented to support TCP MD5, although use of MKTs is not described in RFC 2385.

It is possible to require TCP-AO for a connection or TCP MD5, but it is not possible to require 'either'. When an endpoint is configured to require TCP MD5 for a connection, it must be added to all outgoing segments and validated on all incoming segments [RFC2385]. TCP MD5's requirements prohibit the speculative use of both options for a given connection, e.g., to be decided by the other end of the connection.

9. Interactions with Middleboxes

TCP-AO may interact with middleboxes, depending on their behavior [RFC3234]. Some middleboxes either alter TCP options (such as TCP-AO) directly or alter the information TCP-AO includes in its MAC calculation. TCP-AO may interfere with these devices, exactly where the device modifies information TCP-AO is designed to protect.

9.1. Interactions with Non-NAT/NAPT Middleboxes

TCP-AO supports middleboxes that do not change the IP addresses or ports of segments. Such middleboxes may modify some TCP options, in which case TCP-AO would need to be configured to ignore all options in the MAC calculation on connections traversing that element.

Note that ignoring TCP options may provide less protection, i.e., TCP options could be modified in transit, and such modifications could be used by an attacker. Depending on the modifications, TCP could have compromised efficiency (e.g., timestamp changes), or could cease correct operation (e.g., window scale changes). These vulnerabilities affect only the TCP connections for which TCP-AO is configured to ignore TCP options.

9.2. Interactions with NAT/NAPT Devices

TCP-AO cannot interoperate natively across NAT/NAPT (Network Address Port Translation) devices, which modify the IP addresses and/or port numbers. We anticipate that traversing such devices may require variants of existing NAT/NAPT traversal mechanisms, e.g., encapsulation of the TCP-AO-protected segment in another transport segment (e.g., UDP), as is done in IPsec [RFC2663][RFC3947]. Such variants can be adapted for use with TCP-AO, or IPsec with NAT traversal can be used instead of TCP-AO in such cases [RFC3947].

An alternate proposal for accommodating NATs extends TCP-AO independently of this specification [To10].

10. Evaluation of Requirements Satisfaction

TCP-AO satisfies all the current requirements for a revision to TCP MD5, as summarized below [Ed07].

1. Protected Elements

   A solution to revising TCP MD5 should protect (authenticate) the following elements.

   This is supported -- see Section 5.1.

   a. IP pseudoheader, including IPv4 and IPv6 versions.

      Note that optional coverage is not allowed because IP addresses define a connection. If they can be coordinated across a NAT/NAPT, the sender can compute the MAC based on the received values; if not, a tunnel is required, as noted in Section 9.2.

b. TCP header.

Note that optional port coverage is not allowed because ports define a connection. If they can be coordinated across a NAT/NAPT, the sender can compute the MAC based on the received values; if not, a tunnel is required, as noted in Section 9.2.

c. TCP options.

Note that TCP-AO allows the exclusion of TCP options from coverage, to enable use with middleboxes that modify options (except when they modify TCP-AO itself). See Section 9.

d. TCP payload data.

2. Option Structure Requirements

A solution to revising TCP MD5 should use an option with the following structural requirements.

This is supported -- see Section 5.1.

a. Privacy.

The option should not unnecessarily expose information about the TCP-AO mechanism. The additional protection afforded by keeping this information private may be of little value, but also helps keep the option size small.

TCP-AO exposes only the MKT IDs, MAC, and overall option length on the wire. Note that short MACs could be obscured by using longer option lengths but specifying a short MAC length (this is equivalent to a different MAC algorithm, and is specified in the MKT). See Section 2.2.

b. Allow optional per connection.

The option should not be required on every connection; it should be optional on a per-connection basis.

This is supported because the set of MKTs can be installed to match some connections and not others. Connections not matching any MKT do not require TCP-AO. Further, incoming segments with TCP-AO are not discarded solely because they include the option, provided they do not match any MKT.

c. Require non-optional.

The option should be able to be specified as required for a given connection.

This is supported because the set of MKTs can be installed to match some connections and not others. Connections matching any MKT require TCP-AO.

d. Standard parsing.

The option should be easily parseable, i.e., without conditional parsing, and follow the standard RFC 793 option format.

This is supported -- see Section 2.2.

e. Compatible with Large Windows and SACK.

The option should be compatible with the use of the Large Windows and SACK options.

This is supported -- see Section 7.6. The size of the option is intended to allow use with Large Windows and SACK. See also Section 1.3, which indicates that TCP-AO is 2 bytes shorter than TCP MD5 in the default case, assuming a 96-bit MAC.

3. Cryptography requirements

A solution to revising TCP MD5 should support modern cryptography capabilities.

a. Baseline defaults.

The option should have a default that is required in all implementations.

TCP-AO uses a default required algorithm as specified in [RFC5926] and as noted in Section 5.1 of this document.

b. Good algorithms.

The option should use algorithms considered accepted by the security community, which are considered appropriately safe. The use of non-standard or unpublished algorithms should be avoided.

TCP-AO uses MACs as indicated in [RFC5926]. The KDF is also specified in [RFC5926]. The KDF input string follows the typical design (see [RFC5926]).

c. Algorithm agility.

The option should support algorithms other than the default, to allow agility over time.

TCP-AO allows any desired algorithm, subject to TCP option space limitations, as noted in Section 2.2. The use of a set of MKTs allows separate connections to use different algorithms, both for the MAC and the KDF.

d. Order-independent processing.

The option should be processed independently of the proper order, i.e., they should allow processing of TCP segments in the order received, without requiring reordering. This avoids the need for reordering prior to processing, and avoids the impact of misordered segments on the option.

This is supported -- see Sections 7.3, 7.4, and 7.5. Note that pre-TCP processing is further required, because TCP segments cannot be discarded solely based on a combination of connection state and out-of-window checks; many such segments, although discarded, cause a host to respond with a replay of the last valid ACK, e.g., [RFC793]. See also the derivation of the SNE, which is reconstituted at the receiver using a demonstration algorithm that avoids the need for reordering (in Section 6.2).

e. Security parameter changes require key changes.

The option should require that the MKT change whenever the security parameters change. This avoids the need for coordinating option state during a connection, which is typical for TCP options. This also helps allow "bump in the stack" implementations that are not integrated with endpoint TCP implementations.

Parameters change only when a new MKT is used. See Section 3.

4. Keying requirements.

A solution to revising TCP MD5 should support manual keying, and should support the use of an external automated key management system (e.g., a protocol or other mechanism).

Note that TCP-AO does not specify an MKT management system.

a. Intraconnection rekeying.

   The option should support rekeying during a connection, to avoid the impact of long-duration connections.

   This is supported by the use of IDs and multiple MKTs; see Section 3.

b. Efficient rekeying.

   The option should support rekeying during a connection without the need to expend undue computational resources. In particular, the options should avoid the need to try multiple keys on a given segment.

   This is supported by the use of the KeyID. See Section 6.1.

c. Automated and manual keying.

   The option should support both automated and manual keying.

   The use of MKTs allows external automated and manual keying. See Section 3. This capability is enhanced by the generation of unique per-connection keys, which enables use of manual MKTs with automatically generated traffic keys as noted in Section 5.2.

d. Key management agnostic.

   The option should not assume or require a particular key management solution.

   This is supported by use of a set of MKTs. See Section 3.

5. Expected Constraints

A solution to revising TCP MD5 should also abide by typical safe security practices.

a. Silent failure.

   Receipt of segments failing authentication must result in no visible external action and must not modify internal state, and those events should be logged.

   This is supported - see Sections 7.3, 7.4, and 7.5.

b. At most one such option per segment.

   Only one authentication option can be permitted per segment.

   This is supported by the protocol requirements - see Section 2.2.

c. Outgoing all or none.

   Segments out of a TCP connection are either all authenticated or all not authenticated.

   This is supported - see Section 7.4.

d. Incoming all checked.

   Segments into a TCP connection are always checked to determine whether their authentication should be present and valid.

   This is supported - see Section 7.5.

e. Non-interaction with TCP MD5.

   The use of this option for a given connection should not preclude the use of TCP MD5, e.g., for legacy use, for other connections.

   This is supported - see Section 8.

f. "Hard" ICMP discard.

   The option should allow certain ICMPs to be discarded, notably Type 3 (destination unreachable), Codes 2-4 (transport protocol unreachable, port unreachable, or fragmentation needed and IP DF field set), i.e., the ones indicating the failure of the endpoint to communicate.

   This is supported - see Section 7.8.

g. Maintain TCP connection semantics, in which the socket pair alone defines a TCP association and all its security parameters.

   This is supported - see Sections 3 and 9.

11. Security Considerations

Use of TCP-AO, like the use of TCP MD5 or IPsec, will impact host performance. Connections that are known to use TCP-AO can be attacked by transmitting segments with invalid MACs. Attackers would need to know only the TCP connection ID and TCP-AO Length value to substantially impact the host's processing capacity. This is similar to the susceptibility of IPsec to on-path attacks, where the IP addresses and SPI would be visible. For IPsec, the entire SPI space (32 bits) is arbitrary, whereas for routing protocols typically only the source port (16 bits) is arbitrary (typically with less than 16 bits of randomness [La10]). As a result, it would be easier for an off-path attacker to spoof a TCP-AO segment that could cause receiver validation effort. However, we note that between Internet routers, both ports could be arbitrary (i.e., determined a priori out of band), which would constitute roughly the same off-path antispoofing protection of an arbitrary SPI.

TCP-AO, like TCP MD5, may inhibit connectionless resets. Such resets typically occur after peer crashes, either in response to new connection attempts or when data is sent on stale connections; in either case, the recovering endpoint may lack the connection key required (e.g., if lost during the crash). This may result in timeouts, rather than a more responsive recovery after such a crash. Recommendations for mitigating this effect are discussed in Section 7.7.

TCP-AO does not include a fast decline capability, e.g., where a SYN-ACK is received without an expected TCP-AO and the connection is quickly reset or aborted. Normal TCP operation will retry and timeout, which is what should be expected when the intended receiver is not capable of the TCP variant required anyway. Backoff is not optimized because it would present an opportunity for attackers on the wire to abort authenticated connection attempts by sending spoofed SYN-ACKs without TCP-AO.

TCP-AO is intended to provide similar protections to IPsec, but is not intended to replace the use of IPsec or IKE either for more robust security or more sophisticated security management. TCP-AO is intended to protect the TCP protocol itself from attacks that TLS, sBGP/soBGP, and other data stream protection mechanisms cannot. Like IPsec, TCP-AO does not address the overall issue of ICMP attacks on TCP, but does limit the impact of ICMPs, as noted in Section 7.8.

TCP-AO includes the TCP connection ID (the socket pair) in the MAC calculation. This prevents different concurrent connections using the same MKT (for whatever reason) from potentially enabling a traffic-crossing attack, in which segments to one socket pair are

diverted to attack a different socket pair. When multiple connections use the same MKT, it would be useful to know that segments intended for one ID could not be (maliciously or otherwise) modified in transit and end up being authenticated for the other ID. That requirement would place an additional burden of uniqueness on MKTs within endsystems, and potentially across endsystems. Although the resulting attack is low probability, the protection afforded by including the received ID warrants its inclusion in the MAC, and does not unduly increase the MAC calculation or MKT management.

The use of any security algorithm can present an opportunity for a CPU Denial-of-Service (DoS) attack, where the attacker sends false, random segments that the receiver under attack expends substantial CPU effort to reject. In IPsec, such attacks are reduced by the use of a large Security Parameter Index (SPI) and Sequence Number fields to partly validate segments before CPU cycles are invested validated the Integrity Check Value (ICV). In TCP-AO, the socket pair performs most of the function of IPsec's SPI, and IPsec's Sequence Number, used to avoid replay attacks, isn't needed due to TCP's Sequence Number, which is used to reorder received segments (provided the sequence number doesn't wrap around, which is why TCP-AO adds the SNE in Section 6.2). TCP already protects itself from replays of authentic segment data as well as authentic explicit TCP control (e.g., SYN, FIN, ACK bits) but even authentic replays could affect TCP congestion control [Sa99]. TCP-AO does not protect TCP congestion control from this last form of attack due to the cumbersome nature of layering a windowed security sequence number within TCP in addition to TCP's own sequence number; when such protection is desired, users are encouraged to apply IPsec instead.

Further, it is not useful to validate TCP's Sequence Number before performing a TCP-AO authentication calculation, because out-of-window segments can still cause valid TCP protocol actions (e.g., ACK retransmission) [RFC793]. It is similarly not useful to add a separate Sequence Number field to TCP-AO, because doing so could cause a change in TCP's behavior even when segments are valid.

12. IANA Considerations

The TCP Authentication Option (TCP-AO) was assigned TCP option 29 by IANA action.

This document defines no new namespaces.

To specify MAC and KDF algorithms, TCP-AO refers to a separate document [RFC5926].

13. References

13.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[RFC2403] Madson, C. and R. Glenn, "The Use of HMAC-MD5-96 within ESP and AH", RFC 2403, November 1998.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.

[RFC4306] Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4724] Sangli, S., Chen, E., Fernando, R., Scudder, J., and Y. Rekhter, "Graceful Restart Mechanism for BGP", RFC 4724, January 2007.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.

[RFC4781] Rekhter, Y. and R. Aggarwal, "Graceful Restart Mechanism for BGP with MPLS", RFC 4781, January 2007.

[RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.

13.2. Informative References

[Ba10] Bashyam, M., Jethanandani, M., and A. Ramaiah "Clarification of sender behaviour in persist condition", Work in Progress, January 2010.

[Bo07] Bonica, R., Weis, B., Viswanathan, S., Lange, A., and O. Wheeler, "Authentication for TCP-based Routing and Management Protocols", Work in Progress, February 2007.

[Bo09] Borman, D., "TCP Options and MSS", Work in Progress, July 2009.

[Ed07] Eddy, W., Ed., Bellovin, S., Touch, J., and R. Bonica, "Problem Statement and Requirements for a TCP Authentication Option", Work in Progress, July 2007.

[Go10] Gont, F., "ICMP Attacks against TCP", Work in Progress, March 2010.

[La10] Larsen, M. and F. Gont, "Transport Protocol Port Randomization Recommendations", Work in Progress, April 2010.

[Le09] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", Work in Progress, October 2009.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, May 1996.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[RFC3562] Leech, M., "Key Management Considerations for the TCP MD5 Signature Option", RFC 3562, July 2003.

[RFC3947] Kivinen, T., Swander, B., Huttunen, A., and V. Volpe, "Negotiation of NAT-Traversal in the IKE", RFC 3947, January 2005.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4808] Bellovin, S., "Key Change Strategies for TCP-MD5", RFC 4808, March 2007.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[Sa99] Savage, S., N. Cardwell, D. Wetherall, T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Communications Review, V29, N5, pp71-78, October 1999.

[SDNS88] Secure Data Network Systems, "Security Protocol 4 (SP4)", Specification SDN.401, Revision 1.2, July 12, 1988.

[To07] Touch, J. and A. Mankin, "The TCP Simple Authentication Option", Work in Progress, July 2007.

[To10] Touch, J., "A TCP Authentication Option NAT Extension", Work in Progress, January 2010.

[Wa05] Wang, X., H. Yu, "How to break MD5 and other hash functions", Proc. IACR Eurocrypt 2005, Denmark, pp.19-35.

[We05] Weis, B., Appanna, C., McGrew, D., and A. Ramaiah, "TCP Message Authentication Code Option", Work in Progress, December 2005.

14. Acknowledgments

This document evolved as the result of collaboration of the TCP Authentication Design team (tcp-auth-dt), whose members were (alphabetically): Mark Allman, Steve Bellovin, Ron Bonica, Wes Eddy, Lars Eggert, Charlie Kaufman, Andrew Lange, Allison Mankin, Sandy Murphy, Joe Touch, Sriram Viswanathan, Brian Weis, and Magnus Westerlund. The text of this document is derived from a proposal by Joe Touch and Allison Mankin [To07] (originally from June 2006), which was both inspired by and intended as a counterproposal to the revisions to TCP MD5 suggested in a document by Ron Bonica, Brian Weis, Sriran Viswanathan, Andrew Lange, and Owen Wheeler [Bo07] (originally from September 2005) and in a document by Brian Weis [We05].

Russ Housley suggested L4/application layer management of the master key tuples. Steve Bellovin motivated the KeyID field. Eric Rescorla suggested the use of TCP's Initial Sequence Numbers (ISNs) in the traffic key computation and SNEs to avoid replay attacks, and Brian Weis extended the computation to incorporate the entire connection ID and provided the details of the traffic key computation. Mark Allman, Wes Eddy, Lars Eggert, Ted Faber, Russ Housley, Gregory Lebovitz, Tim Polk, Eric Rescorla, Joe Touch, and Brian Weis developed the master key coordination mechanism.

Alfred Hoenes, Charlie Kaufman, Adam Langley, and numerous other members of the TCPM WG also provided substantial feedback on this document.

This document was originally prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695
U.S.A.

Phone: +1 (310) 448-9151
EMail: touch@isi.edu
URL: http://www.isi.edu/touch

Allison Mankin
Johns Hopkins Univ.
Baltimore, MD
U.S.A.

Phone: 1 301 728 7199
EMail: mankin@psg.com
URL: http://www.psg.com/~mankin/

Ronald P. Bonica
Juniper Networks
2251 Corporate Park Drive
Herndon, VA 20171
U.S.A.

EMail: rbonica@juniper.net