Internet Engineering Task Force (IETF)
Request for Comments: 8547
Category: Experimental
ISSN: 2070-1721

A. Bittau
Google
D. Giffin
Stanford University
M. Handley
University College London
D. Mazieres
Stanford University
E. Smith
Kestrel Institute

May 2019

TCP-ENO: Encryption Negotiation Option

Abstract

Despite growing adoption of TLS, a significant fraction of TCP traffic on the Internet remains unencrypted. The persistence of unencrypted traffic can be attributed to at least two factors. First, some legacy protocols lack a signaling mechanism (such as a STARTTLS command) by which to convey support for encryption, thus making incremental deployment impossible. Second, legacy applications themselves cannot always be upgraded and therefore require a way to implement encryption transparently entirely within the transport layer. The TCP Encryption Negotiation Option (TCP-ENO) addresses both of these problems through a new TCP option kind providing out-of-band, fully backward-compatible negotiation of encryption.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. 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). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8547.

Table of Contents

1. Introduction

Many applications and protocols running on top of TCP today do not encrypt traffic. This failure to encrypt lowers the bar for certain attacks, harming both user privacy and system security. Counteracting the problem demands a minimally intrusive, backward-compatible mechanism for incrementally deploying encryption. The TCP Encryption Negotiation Option (TCP-ENO) specified in this document provides such a mechanism.

Introducing TCP options, extending operating system interfaces to support TCP-level encryption, and extending applications to take advantage of TCP-level encryption all require effort. To the greatest extent possible, the effort invested in realizing TCP-level encryption today needs to remain applicable in the future should the need arise to change encryption strategies. To this end, it is useful to consider two questions separately:

1. How to negotiate the use of encryption at the TCP layer

2. How to perform encryption at the TCP layer

This document addresses question 1 with a new TCP option, ENO. TCP-ENO provides a framework in which two endpoints can agree on a TCP encryption protocol (TEP) out of multiple possible TEPs. For future compatibility, TEPs can vary widely in terms of wire format, use of TCP option space, and integration with the TCP header and segmentation. However, ENO abstracts these differences to ensure the introduction of new TEPs can be transparent to applications taking advantage of TCP-level encryption.

Question 2 is addressed by one or more companion TEP specification documents. While current TEPs enable TCP-level traffic encryption today, TCP-ENO ensures that the effort invested to deploy today's TEPs will additionally benefit future ones.

1.1. Design Goals

TCP-ENO was designed to achieve the following goals:

1. Enable endpoints to negotiate the use of a separately specified TCP encryption protocol (TEP) suitable for either opportunistic security [RFC7435] of arbitrary TCP communications or stronger security of applications willing to perform endpoint authentication.

2. Transparently fall back to unencrypted TCP when not supported by both endpoints.

3. Provide out-of-band signaling through which applications can better take advantage of TCP-level encryption (for instance, by improving authentication mechanisms in the presence of TCP-level encryption).

4. Define a standard negotiation transcript that TEPs can use to defend against tampering with TCP-ENO.

5. Make parsimonious use of TCP option space.

6. Define roles for the two ends of a TCP connection, so as to name each end of a connection for encryption or authentication purposes even following a symmetric simultaneous open.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Throughout this document, we use the following terms, several of which have more detailed normative descriptions in [RFC793]:

SYN segment
: A TCP segment in which the SYN flag is set

ACK segment
: A TCP segment in which the ACK flag is set (which includes most segments other than an initial SYN segment)

Non-SYN segment
: A TCP segment in which the SYN flag is clear

SYN-only segment
: A TCP segment in which the SYN flag is set but the ACK flag is clear

SYN-ACK segment
: A TCP segment in which the SYN and ACK flags are both set

Active opener
: A host that initiates a connection by sending a SYN-only segment. With the BSD socket API, an active opener calls "connect". In client-server configurations, active openers are typically clients.

Passive opener
: A host that does not send a SYN-only segment but responds to one with a SYN-ACK segment. With the BSD socket API, passive openers call "listen" and "accept", rather than "connect". In client-server configurations, passive openers are typically servers.

Simultaneous open
: The act of symmetrically establishing a TCP connection between two active openers (both of which call "connect" with BSD sockets). Each host of a simultaneous open sends both a SYN-only and a SYN-ACK segment. Simultaneous open is less common than asymmetric open with one active and one passive opener, but it can be used for NAT traversal by peer-to-peer applications [RFC5382].

TEP
: A TCP encryption protocol intended for use with TCP-ENO and specified in a separate document

TEP identifier
: A unique 7-bit value in the range 0x20-0x7f that IANA has assigned to a TEP

Negotiated TEP
: The single TEP governing a TCP connection, determined by use of the TCP ENO option specified in this document

4. TCP-ENO Specification

TCP-ENO extends TCP connection establishment to enable encryption opportunistically. It uses a new TCP option kind [RFC793] to negotiate one among multiple possible TCP encryption protocols (TEPs). The negotiation involves hosts exchanging sets of supported TEPs, where each TEP is represented by a suboption within a larger TCP ENO option in the offering host’s SYN segment.

If TCP-ENO succeeds, it yields the following information:

- a negotiated TEP represented by a unique 7-bit TEP identifier,
- a few extra bytes of suboption data from each host, if needed by the TEP,

- a negotiation transcript with which to mitigate attacks on the negotiation itself,
- role assignments designating one endpoint "host A" and the other endpoint "host B", and
- a bit available to higher-layer protocols at each endpoint for out-of-band negotiation of updated behavior in the presence of TCP encryption.

If TCP-ENO fails, encryption is disabled and the connection falls back to traditional unencrypted TCP.

The remainder of this section provides the normative description of the TCP ENO option and handshake protocol.

4.1. ENO Option

TCP-ENO employs an option in the TCP header [RFC793]. Figure 1 illustrates the high-level format of this option.

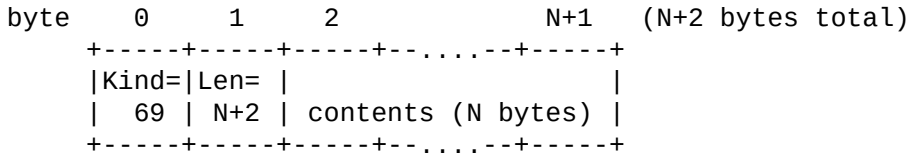


Figure 1: The TCP-ENO Option

The contents of an ENO option can take one of two forms. A SYN-form ENO option, illustrated in Figure 2, appears only in SYN segments. A non-SYN-form ENO option, illustrated in Figure 3, appears only in non-SYN segments. The SYN-form ENO option acts as a container for zero or more suboptions, labeled "Opt_0", "Opt_1", ... in Figure 2. The non-SYN-form ENO option, by its presence, acts as a one-bit acknowledgment, with the actual contents ignored by ENO. Particular TEPs MAY assign additional meaning to the contents of non-SYN-form ENO options. When a negotiated TEP does not assign such meaning, the contents of a non-SYN-form ENO option MUST be zero bytes (i.e., N = 0) in sent segments and MUST be ignored in received segments.

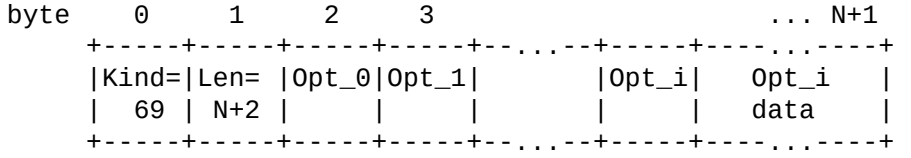


Figure 2: SYN-Form ENO Option

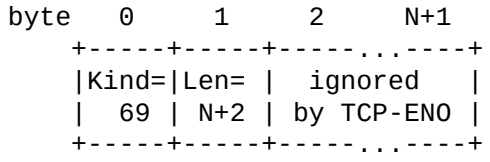
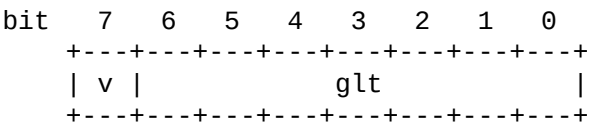


Figure 3: Non-SYN-Form ENO option, Where N MAY Be 0

Every suboption starts with a byte of the form illustrated in Figure 4. The high bit "v", when set, introduces suboptions with variable-length data. When v = 0, the byte itself constitutes the entirety of the suboption. The remaining 7-bit value, called "glt", takes on various meanings as defined below:

- Global configuration data (discussed in Section 4.2)
- Suboption data length for the next suboption (discussed in Section 4.4)
- An offer to use a particular TEP defined in a separate TEP specification document



v - non-zero for use with variable-length suboption data
glt - Global suboption, Length, or TEP identifier

Figure 4: Format of Initial Suboption Byte

Table 1 summarizes the meaning of initial suboption bytes. Values of glt below 0x20 are used for global suboptions and length information (the "gl" in "glt"), while those greater than or equal to 0x20 are TEP identifiers (the "t"). When v = 0, since the initial suboption byte constitutes the entirety of the suboption, all information is expressed by the 7-bit glt value, which can be either a global suboption or a TEP identifier. When v = 1, it indicates a suboption with variable-length suboption data. Only TEP identifiers have suboption data, not global suboptions. Therefore, bytes with v = 1 and glt < 0x20 are not global suboptions but rather length bytes governing the length of the next suboption (which MUST be a TEP identifier). In the absence of a length byte, a TEP identifier suboption with v = 1 has suboption data extending to the end of the TCP option.

glt	v	Meaning
0x00-0x1f	0	Global suboption (Section 4.2)
0x00-0x1f	1	Length byte (Section 4.4)
0x20-0x7f	0	TEP identifier without suboption data
0x20-0x7f	1	TEP identifier followed by suboption data

Table 1: Initial Suboption Byte Values

A SYN segment MUST contain at most one TCP ENO option. If a SYN segment contains more than one ENO option, the receiver MUST behave as though the segment contained no ENO options and disable encryption. A TEP MAY specify the use of multiple ENO options in a non-SYN segment. For non-SYN segments, ENO itself only distinguishes between the presence or absence of ENO options; multiple ENO options are interpreted the same as one.

4.2. The Global Suboption

Suboptions 0x00-0x1f are used for global configuration that applies regardless of the negotiated TEP. A TCP SYN segment MUST include at most one ENO suboption in this range. A receiver MUST ignore all but the first suboption in this range in any given TCP segment so as to anticipate updates to ENO that assign new meaning to bits in subsequent global suboptions. The value of a global suboption byte is interpreted as a bit mask, illustrated in Figure 5.

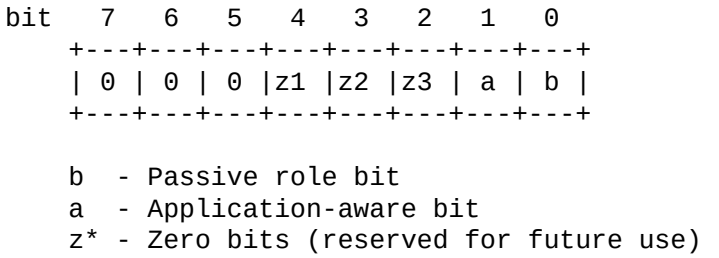


Figure 5: Format of the Global Suboption Byte

The fields of the bit mask are interpreted as follows:

b
: The passive role bit MUST be 1 for all passive openers. For active openers, it MUST default to 0, but implementations MUST provide an API through which an application can explicitly set b = 1 before initiating an active open. (Manual configuration of "b" is only necessary to enable encryption with a simultaneous open

and requires prior coordination to ensure exactly one endpoint sets b = 1 before connecting.) See Section 8.3 for further discussion.

a
: Legacy applications can benefit from ENO-specific updates that improve endpoint authentication or avoid double encryption. The application-aware bit "a" is an out-of-band signal through which higher-layer protocols can enable ENO-specific updates that would otherwise not be backwards compatible. Implementations MUST set this bit to zero by default, and MUST provide an API through which applications can change the value of the bit as well as examine the value of the bit sent by the remote host. Implementations MUST furthermore support a mandatory application-aware mode in which TCP-ENO is automatically disabled if the remote host does not set a = 1. See Section 8.4 for further discussion.

z1, z2, z3
: The "z" bits are reserved for future updates to TCP-ENO. They MUST be set to zero in sent segments and MUST be ignored in received segments.

A SYN segment without an explicit global suboption has an implicit global suboption of 0x00. Because passive openers MUST always set b = 1, they cannot rely on this implicit 0x00 byte and MUST include an explicit global suboption in their SYN-ACK segments.

4.3. TCP-ENO Roles

TCP-ENO uses abstract roles called "A" and "B" to distinguish the two ends of a TCP connection. These roles are determined by the "b" bit in the global suboption. The host that sent an implicit or explicit suboption with b = 0 plays the A role. The host that sent b = 1 plays the B role. Because a passive opener MUST set b = 1 and an active opener by default has b = 0, the normal case is for the active opener to play role A and the passive opener role B.

Applications performing a simultaneous open, if they desire TCP-level encryption, need to arrange for exactly one endpoint to set b = 1 (despite being an active opener) while the other endpoint keeps the default b = 0. Otherwise, if both sides use the default b = 0 or if both sides set b = 1, then TCP-ENO will fail and fall back to unencrypted TCP. Likewise, if an active opener explicitly configures b = 1 and connects to a passive opener (which MUST always have b = 1), then TCP-ENO will fail and fall back to unencrypted TCP.

TEP specifications SHOULD refer to TCP-ENO's A and B roles to specify asymmetric behavior by the two hosts. For the remainder of this document, we will use the terms "host A" and "host B" to designate the hosts with roles A and B, respectively, in a connection.

4.4. Specifying Suboption Data Length

A TEP MAY optionally make use of one or more bytes of suboption data. The presence of such data is indicated by setting v = 1 in the initial suboption byte (see Figure 4). A suboption introduced by a TEP identifier with v = 1 (i.e., a suboption whose first octet has value 0xa0 or higher) extends to the end of the TCP option. Hence, if only one suboption requires data, the most compact way to encode it is to place it last in the ENO option, after all other suboptions. In Figure 2, for example, the last suboption, Opt_i, has suboption data and thus requires v = 1. However, the suboption data length is inferred from the total length of the TCP option.

When a suboption with data is not last in an ENO option, the sender MUST explicitly specify the suboption data length for the receiver to know where the next suboption starts. The sender does so by introducing the suboption with a length byte, depicted in Figure 6. The length byte encodes a 5-bit value nnnnn. Adding one to nnnnn yields the length of the suboption data (not including the length byte or the TEP identifier). Hence, a length byte can designate anywhere from 1 to 32 bytes of suboption data (inclusive).

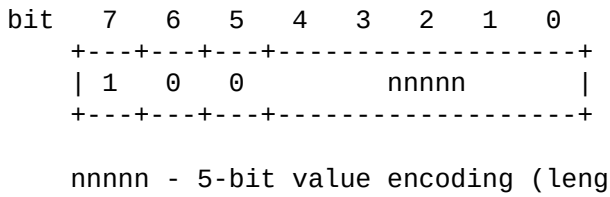


Figure 6: Format of a Length Byte

A suboption preceded by a length byte MUST be a TEP identifier (glt >= 0x20) and MUST have v = 1. Figure 7 shows an example of such a suboption.

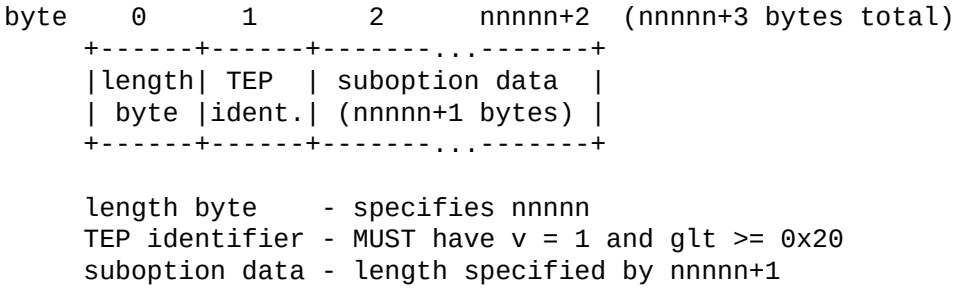


Figure 7: Suboption with Length Byte

A host MUST ignore an ENO option in a SYN segment and MUST disable encryption if either of the following apply:

1. A length byte indicates that suboption data would extend beyond the end of the TCP ENO option.

2. A length byte is followed by an octet in the range 0x00-0x9f (meaning the following byte has v = 0 or glt < 0x20).

Because the last suboption in an ENO option is special-cased to have its length inferred from the 8-bit TCP option length, it MAY contain more than 32 bytes of suboption data. Other suboptions are limited to 32 bytes by the length byte format. However, the TCP header itself can only accommodate a maximum of 40 bytes of options. Therefore, regardless of the length byte format, a segment would not be able to contain more than one suboption over 32 bytes in size. That said, TEPs MAY define the use of multiple suboptions with the same TEP identifier in the same SYN segment, providing another way to convey over 32 bytes of suboption data even with length bytes.

4.5. The Negotiated TEP

A TEP identifier glt (with glt >= 0x20) is valid for a connection when all of the following hold:

1. Each side has sent a suboption for glt in its SYN-form ENO option.

2. Any suboption data in these glt suboptions is valid according to the TEP specification and satisfies any runtime constraints.

3. If an ENO option contains multiple suboptions with glt, then such repetition is well-defined by the TEP specification.

A passive opener (which is always host B) sees the remote host's SYN segment before constructing its own SYN-ACK segment. Therefore, a passive opener SHOULD include only one TEP identifier in SYN-ACK segments and SHOULD ensure this TEP identifier is valid. However, simultaneous open or implementation considerations can prevent host B from offering only one TEP.

To accommodate scenarios in which host B sends multiple TEP identifiers in the SYN-ACK segment, the negotiated TEP is defined as the last valid TEP identifier in host B's SYN-form ENO option. This definition means host B specifies TEP suboptions in order of increasing priority, while host A does not influence TEP priority.

4.6. TCP-ENO Handshake

A host employing TCP-ENO for a connection MUST include an ENO option in every TCP segment sent until either encryption is disabled or the host receives a non-SYN segment. In particular, this means an active opener MUST include a non-SYN-form ENO option in the third segment of a three-way handshake.

A host MUST disable encryption, refrain from sending any further ENO options, and fall back to unencrypted TCP if any of the following occurs:

1. Any segment it receives up to and including the first received ACK segment does not contain an ENO option (or contains an ill-formed SYN-form ENO option).

2. The SYN segment it receives does not contain a valid TEP identifier.

3. It receives a SYN segment with an incompatible global suboption. (Specifically, "incompatible" means the two hosts set the same "b" value, or the connection is in mandatory application-aware mode and the remote host set a = 0.)

Hosts MUST NOT alter SYN-form ENO options in retransmitted segments, or between the SYN and SYN-ACK segments of a simultaneous open, with two exceptions for an active opener. First, an active opener MAY unilaterally disable ENO (and thus remove the ENO option) between retransmissions of a SYN-only segment. (Such removal could enable recovery from middleboxes dropping segments with ENO options.) Second, an active opener performing simultaneous open MAY include no TCP-ENO option in its SYN-ACK if the received SYN caused it to disable encryption according to the above rules (for instance, because role negotiation failed).

Once a host has both sent and received an ACK segment containing an ENO option, encryption MUST be enabled. Once encryption is enabled, hosts MUST follow the specification of the negotiated TEP and MUST NOT present raw TCP payload data to the application. In particular, data segments MUST NOT contain plaintext application data, but rather ciphertext, key negotiation parameters, or other messages as determined by the negotiated TEP.

A host MAY send a SYN-form ENO option containing zero TEP identifier suboptions, which we term a "vacuous" ENO option. If either host's SYN segment contains a vacuous ENO option, it follows that there are no valid TEP identifiers for the connection, and therefore the connection MUST fall back to unencrypted TCP. Hosts MAY send vacuous ENO options to indicate that ENO is supported but unavailable by configuration, or to probe network paths for robustness to ENO options. However, a passive opener MUST NOT send a vacuous ENO option in a SYN-ACK segment unless there was an ENO option in the SYN segment it received. Moreover, a passive opener's SYN-form ENO option MUST still include a global suboption with b = 1 as discussed in Section 4.3.

4.7. Data in SYN Segments

TEPs MAY specify the use of data in SYN segments so as to reduce the number of round trips required for connection setup. The meaning of data in a SYN segment with an ENO option (a SYN+ENO segment) is determined by the last TEP identifier in the ENO option, which we term the segment's "SYN TEP". A SYN+ENO segment MAY of course include multiple TEP suboptions, but only the SYN TEP (i.e., the last one) specifies how to interpret the SYN segment's data payload.

A host sending a SYN+ENO segment MUST NOT include data in the segment unless the SYN TEP's specification defines the use of such data. Furthermore, to avoid conflicting interpretations of SYN data, a SYN+ENO segment MUST NOT include a non-empty TCP Fast Open (TFO) option [RFC7413].

Because a host can send SYN data before knowing which if any TEP the connection will negotiate, hosts implementing ENO are REQUIRED to discard data from SYN+ENO segments when the SYN TEP does not become the negotiated TEP. Hosts are furthermore REQUIRED to discard SYN data in cases where another Internet standard specifies a conflicting interpretation of SYN data (as would occur when receiving a non-empty TFO option). This requirement applies to hosts that implement ENO even when ENO has been disabled by configuration. However, note that discarding SYN data is already common practice [RFC4987] and the new requirement applies only to segments containing ENO options.

More specifically, a host that implements ENO MUST discard the data in a received SYN+ENO segment if any of the following applies:

- ENO fails and TEP-indicated encryption is disabled for the connection.
- The received segment's SYN TEP is not the negotiated TEP.
- The negotiated TEP does not define the use of SYN data.
- The SYN segment contains a non-empty TFO option or any other TCP option implying a conflicting definition of SYN data.

A host discarding SYN data in compliance with the above requirement MUST NOT acknowledge the sequence number of the discarded data, but rather MUST acknowledge the other host's initial sequence number as if the received SYN segment contained no data. Furthermore, after discarding SYN data, such a host MUST NOT assume the SYN data will be identically retransmitted, and MUST process data only from non-SYN segments.

If a host sends a SYN+ENO segment with data and receives acknowledgment for the data, but the SYN TEP in its transmitted SYN segment is not the negotiated TEP (either because a different TEP was negotiated or because ENO failed to negotiate encryption), then the host MUST abort the TCP connection. Proceeding in any other fashion risks misinterpreted SYN data.

If a host sends a SYN-only SYN+ENO segment bearing data and subsequently receives a SYN-ACK segment without an ENO option, that host MUST abort the connection even if the SYN-ACK segment does not acknowledge the SYN data. The issue is that unacknowledged data could nonetheless have been cached by the receiver; later retransmissions intended to supersede this unacknowledged data could fail to do so if the receiver gives precedence to the cached original data. Implementations MAY provide an API call for a non-default mode in which unacknowledged SYN data does not cause a connection abort, but applications MUST use this mode only when a higher-layer integrity check would anyway terminate a garbled connection.

To avoid unexpected connection aborts, ENO implementations MUST disable the use of data in SYN-only segments by default. Such data MAY be enabled by an API command. In particular, implementations MAY provide a per-connection mandatory encryption mode that automatically aborts a connection if ENO fails, and they MAY enable SYN data in this mode.

To satisfy the requirement of the previous paragraph, all TEPs SHOULD support a normal mode of operation that avoids data in SYN-only segments. An exception is TEPs intended to be disabled by default.

4.8. Negotiation Transcript

To defend against attacks on encryption negotiation itself, a TEP MUST, with high probability, fail to establish a working connection between two ENO-compliant hosts when SYN-form ENO options have been altered in transit. (Of course, in the absence of endpoint authentication, two compliant hosts can each still be connected to a man-in-the-middle attacker.) To detect SYN-form ENO option tampering, TEPs MUST reference a transcript of TCP-ENO's negotiation.

TCP-ENO defines its negotiation transcript as a packed data structure consisting of two TCP-ENO options exactly as they appeared in the TCP header (including the TCP option kind and TCP option length byte as illustrated in Figure 1). The transcript is constructed from the following, in order:

1. The TCP-ENO option in host A's SYN segment, including the kind and length bytes

2. The TCP-ENO option in host B's SYN segment, including the kind and length bytes

Note that because the ENO options in the transcript contain length bytes as specified by TCP, the transcript unambiguously delimits A's and B's ENO options.

5. Requirements for TEPs

TCP-ENO affords TEP specifications a large amount of design flexibility. However, to abstract TEP differences away from applications requires fitting them all into a coherent framework. As such, any TEP claiming an ENO TEP identifier MUST satisfy the following normative list of properties:

- TEPs MUST protect TCP data streams with authenticated encryption. (Note that "authenticated encryption" refers only to the form of encryption, such as an Authenticated Encryption with Associated Data (AEAD) algorithm meeting the requirements of [RFC5116]; it does not imply endpoint authentication.)

- TEPs MUST define a session ID whose value identifies the TCP connection and, with overwhelming probability, is unique over all time if either host correctly obeys the TEP. Section 5.1 describes the requirements of the session ID in more detail.

- TEPs MUST NOT make data confidentiality dependent on encryption algorithms with a security strength [NIST-SP-800-57] of less than 120 bits. The number 120 was chosen to accommodate ciphers with 128-bit keys that lose a few bits of security either to particularities of the key schedule or to highly theoretical and unrealistic attacks.

- TEPs MUST NOT allow the negotiation of null cipher suites, even for debugging purposes. (Implementations MAY support debugging modes that allow applications to extract their own session keys.)

- TEPs MUST guarantee the confidentiality of TCP streams without assuming the security of any long-lived secrets. Implementations SHOULD provide forward secrecy soon after the close of a TCP connection and SHOULD therefore bound the delay between closing a connection and erasing any relevant cryptographic secrets. (Exceptions to forward secrecy are permissible only at the implementation level and only in response to hardware or architectural constraints -- e.g., storage that cannot be securely erased.)

- TEPs MUST protect and authenticate the end-of-file marker conveyed by TCP's FIN flag. In particular, a receiver MUST, with overwhelming probability, detect a FIN flag that was set or cleared in transit and does not match the sender's intent. A TEP MAY discard a segment with such a corrupted FIN bit or MAY abort the connection in response to such a segment. However, any such abort MUST raise an error condition distinct from an authentic end-of-file condition.

- TEPs MUST prevent corrupted packets from causing urgent data to be delivered when none has been sent. There are several ways to do so. For instance, a TEP MAY cryptographically protect the URG flag and urgent pointer alongside ordinary payload data. Alternatively, a TEP MAY disable urgent data functionality by clearing the URG flag on all received segments and returning errors in response to sender-side urgent-data API calls. Implementations SHOULD avoid negotiating TEPs that disable urgent data by default. The exception is when applications and protocols are known never to send urgent data.

5.1. Session IDs

Each TEP MUST define a session ID that is computable by both endpoints and uniquely identifies each encrypted TCP connection. Implementations MUST expose the session ID to applications via an API extension. The API extension MUST return an error when no session ID is available because ENO has failed to negotiate encryption or because no connection is yet established. Applications that are aware of TCP-ENO SHOULD, when practical, authenticate the TCP endpoints by incorporating the values of the session ID and TCP-ENO role (A or B) into higher-layer authentication mechanisms.

In order to avoid replay attacks and prevent authenticated session IDs from being used out of context, session IDs MUST be unique over all time with high probability. This uniqueness property MUST hold even if one end of a connection maliciously manipulates the protocol in an effort to create duplicate session IDs. In other words, it MUST be infeasible for a host, even by violating the TEP specification, to establish two TCP connections with the same session ID to remote hosts properly implementing the TEP.

To prevent session IDs from being confused across TEPs, all session IDs begin with the negotiated TEP identifier -- that is, the last valid TEP identifier in host B's SYN segment. Furthermore, this initial byte has bit "v" set to the same value that accompanied the negotiated TEP identifier in B's SYN segment. However, only this single byte is included, not any suboption data. Figure 8 shows the resulting format. This format is designed for TEPs to compute unique identifiers; it is not intended for application authors to pick apart session IDs. Applications SHOULD treat session IDs as monolithic opaque values and SHOULD NOT discard the first byte to shorten identifiers. (An exception is for non-security-relevant purposes, such as gathering statistics about negotiated TEPs.)

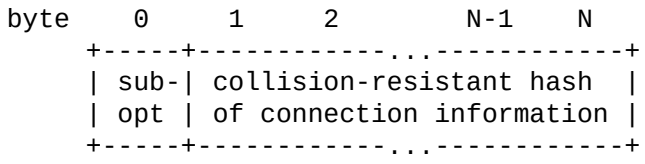


Figure 8: Format of a Session ID

Though TEP specifications retain considerable flexibility in their definitions of the session ID, all session IDs MUST meet the following normative list of requirements:

- The session ID MUST be at least 33 bytes (including the one-byte suboption), though TEPs MAY choose longer session IDs.

- The session ID MUST depend, in a collision-resistant way, on all of the following (meaning it is computationally infeasible to produce collisions of the session ID derivation function unless all of the following quantities are identical):
  - Fresh data contributed by both sides of the connection
  - Any public keys, public Diffie-Hellman parameters, or other public asymmetric cryptographic parameters that are employed by the TEP and have corresponding private data that is known by only one side of the connection
  - The negotiation transcript specified in Section 4.8
- Unless and until applications disclose information about the session ID, all but the first byte MUST be computationally indistinguishable from random bytes to a network eavesdropper.
- Applications MAY choose to make session IDs public. Therefore, TEPs MUST NOT place any confidential data in the session ID (such as data permitting the derivation of session keys).

6. Examples

This subsection illustrates the TCP-ENO handshake with a few non-normative examples.

```
(1) A -> B:  SYN      ENO<X,Y>
(2) B -> A:  SYN-ACK  ENO<b=1,Y>
(3) A -> B:  ACK      ENO<>
[rest of connection encrypted according to TEP Y]
```

Figure 9: Three-Way Handshake with Successful TCP-ENO Negotiation

Figure 9 shows a three-way handshake with a successful TCP-ENO negotiation. Host A includes two ENO suboptions with TEP identifiers X and Y. Host A does not include an explicit global suboption, which means it has an implicit global suboption 0x00 conveying passive role bit b = 0. The two sides agree to follow the TEP identified by suboption Y.

```
(1) A -> B:  SYN      ENO<X,Y>
(2) B -> A:  SYN-ACK
(3) A -> B:  ACK
[rest of connection unencrypted legacy TCP]
```

Figure 10: Three-Way Handshake with Failed TCP-ENO Negotiation

Figure 10 shows a failed TCP-ENO negotiation. The active opener (A) indicates support for TEPs corresponding to suboptions X and Y. Unfortunately, at this point, one of several things occurs:

1. The passive opener (B) does not support TCP-ENO.

2. B supports TCP-ENO but supports neither of the TEPs X and Y, and so it does not reply with an ENO option.

3. B supports TCP-ENO but has the connection configured in mandatory application-aware mode and thus disables ENO because A's SYN segment contains an implicit global suboption with a = 0.

4. The network stripped the ENO option out of A's SYN segment, so B did not receive it.

Whichever of the above applies, the connection transparently falls back to unencrypted TCP.

```
(1) A -> B:  SYN      ENO<X,Y>
(2) B -> A:  SYN-ACK  ENO<b=1,X> [ENO stripped by middlebox]
(3) A -> B:  ACK
[rest of connection unencrypted legacy TCP]
```

Figure 11: Failed TCP-ENO Negotiation Because of Option Stripping

Figure 11 Shows another handshake with a failed encryption negotiation. In this case, the passive opener (B) receives an ENO option from A and replies. However, the reverse network path from B to A strips ENO options. Therefore, A does not receive an ENO option from B, it disables ENO, and it does not include a non-SYN-form ENO option in segment 3 when ACKing B's SYN. Had A not disabled encryption, Section 4.6 would have required it to include a non-SYN-form ENO option in segment 3. The omission of this option informs B that encryption negotiation has failed, after which the two hosts proceed with unencrypted TCP.

```
(1) A -> B:  SYN      ENO<Y,X>
(2) B -> A:  SYN      ENO<b=1,X,Y,Z>
(3) A -> B:  SYN-ACK  ENO<Y,X>
(4) B -> A:  SYN-ACK  ENO<b=1,X,Y,Z>
[rest of connection encrypted according to TEP Y]
```

Figure 12: Simultaneous Open with Successful TCP-ENO Negotiation

Figure 12 shows a successful TCP-ENO negotiation with simultaneous open. Here, the first four segments contain a SYN-form ENO option, as each side sends both a SYN-only and a SYN-ACK segment. The ENO

option in each host's SYN-ACK is identical to the ENO option in its SYN-only segment, as otherwise, connection establishment could not recover from the loss of a SYN segment. The last valid TEP in host B's ENO option is Y, so Y is the negotiated TEP.

7. Future Developments

TCP-ENO is designed to capitalize on future developments that could alter trade-offs and change the best approach to TCP-level encryption (beyond introducing new cipher suites). By way of example, we discuss a few such possible developments.

Various proposals exist to increase the maximum space for options in the TCP header. These proposals are highly experimental -- particularly those that apply to SYN segments. Therefore, future TEPs are unlikely to benefit from extended SYN option space. In the unlikely event that SYN option space is one day extended, however, future TEPs could benefit by embedding key agreement messages directly in SYN segments. Under such usage, the 32-byte limit on length bytes could prove insufficient. This document intentionally aborts TCP-ENO if a length byte is followed by an octet in the range 0x00-0x9f. If necessary, a future update to this document can define a format for larger suboptions by assigning meaning to such currently undefined byte sequences.

New revisions to socket interfaces [RFC3493] could involve library calls that simultaneously have access to hostname information and an underlying TCP connection. Such an API enables the possibility of authenticating servers transparently to the application, particularly in conjunction with technologies such as DNS-Based Authentication of Named Entities (DANE) [RFC6394]. An update to TCP-ENO can adopt one of the "z" bits in the global suboption to negotiate the use of an endpoint authentication protocol before any application use of the TCP connection. Over time, the consequences of failed or missing endpoint authentication can gradually be increased from issuing log messages to aborting the connection if some as yet unspecified DNS record indicates authentication is mandatory. Through shared library updates, such endpoint authentication can potentially be added transparently to legacy applications without recompilation.

TLS can currently only be added to legacy applications whose protocols accommodate a STARTTLS command or equivalent. TCP-ENO, because it provides out-of-band signaling, opens the possibility of future TLS revisions being generically applicable to any TCP application.

8. Design Rationale

This section describes some of the design rationale behind TCP-ENO.

8.1. Handshake Robustness

Incremental deployment of TCP-ENO depends critically on failure cases devolving to unencrypted TCP rather than causing the entire TCP connection to fail.

Because a network path might drop ENO options in one direction only, a host needs to know not just that the peer supports encryption, but that the peer has received an ENO option. To this end, ENO disables encryption unless it receives an ACK segment bearing an ENO option. To stay robust in the face of dropped segments, hosts continue to include non-SYN-form ENO options in segments until the point that they have received a non-SYN segment from the other side.

One particularly pernicious middlebox behavior found in the wild is load balancers that echo unknown TCP options found in SYN segments back to an active opener. The passive role bit "b" in global suboptions ensures encryption will always be disabled under such circumstances, as sending back a verbatim copy of an active opener's SYN-form ENO option always causes role negotiation to fail.

8.2. Suboption Data

TEPs can employ suboption data for session caching, cipher suite negotiation, or other purposes. However, TCP currently limits total option space consumed by all options to only 40 bytes, making it impractical to have many suboptions with data. For this reason, ENO optimizes the case of a single suboption with data by inferring the length of the last suboption from the TCP option length. Doing so saves one byte.

8.3. Passive Role Bit

TCP-ENO, TEPs, and applications all have asymmetries that require an unambiguous way to identify one of the two connection endpoints. As an example, Section 4.8 specifies that host A's ENO option comes before host B's in the negotiation transcript. As another example, an application might need to authenticate one end of a TCP connection with a digital signature. To ensure the signed message cannot be interpreted out of context to authenticate the other end, the signed message would need to include both the session ID and the local role, A or B.

A normal TCP three-way handshake involves one active and one passive opener. This asymmetry is captured by the default configuration of the "b" bit in the global suboption. With simultaneous open, both hosts are active openers, so TCP-ENO requires that one host explicitly configure b = 1. An alternate design might automatically break the symmetry to avoid this need for explicit configuration. However, all such designs we considered either lacked robustness or consumed precious bytes of SYN option space even in the absence of simultaneous open. (One complicating factor is that TCP does not know it is participating in a simultaneous open until after it has sent a SYN segment. Moreover, with packet loss, one host might never learn it has participated in a simultaneous open.)

8.4. Application-Aware Bit

Applications developed before TCP-ENO can potentially evolve to take advantage of TCP-level encryption. For instance, an application designed to run only on trusted networks might leverage TCP-ENO to run on untrusted networks, but, importantly, needs to authenticate endpoints and session IDs to do so. In addition to user-visible changes such as requesting credentials, this kind of authentication functionality requires application-layer protocol changes. Some protocols can accommodate the requisite changes -- for instance, by introducing a new verb analogous to STARTTLS, while others cannot do so in a backwards-compatible manner.

The application-aware bit "a" in the global suboption provides a means of incrementally deploying enhancements specific to TCP-ENO to application-layer protocols that would otherwise lack the necessary extensibility. Software implementing the enhancement always sets a = 1 in its own global suboption, but only activates the new behavior when the other end of the connection also sets a = 1.

A related issue is that an application might leverage TCP-ENO as a replacement for legacy application-layer encryption. In this scenario, if both endpoints support TCP-ENO, then application-layer encryption can be disabled in favor of simply authenticating the TCP-ENO session ID. On the other hand, if one endpoint is not aware of the new mode of operation specific to TCP-ENO, there is little benefit to performing redundant encryption at the TCP layer; data is already encrypted once at the application layer, and authentication only has meaning with respect to this application-layer encryption. The mandatory application-aware mode lets applications avoid double encryption in this case: the mode sets a = 1 in the local host’s global suboption but also disables TCP-ENO entirely in the event that the other side has not also set a = 1.

Note that the application-aware bit is not needed by applications that already support adequate higher-layer encryption such as those provided by TLS [RFC8446] or SSH [RFC4253]. To avoid double encryption in such cases, it suffices to disable TCP-ENO by configuration on any ports with known secure protocols.

8.5. Use of ENO Option Kind by TEPs

This document does not specify the use of ENO options beyond the first few segments of a connection. Moreover, it does not specify the content of ENO options in non-SYN segments, only their presence. As a result, any use of option kind 69 after the SYN exchange does not conflict with this document. In addition, because ENO guarantees at most one negotiated TEP per connection, TEPs will not conflict with one another or ENO if they use option kind 69 for out-of-band signaling in non-SYN segments.

8.6. Unpredictability of Session IDs

Section 5.1 specifies that all but the first (TEP identifier) byte of a session ID MUST be computationally indistinguishable from random bytes to a network eavesdropper. This property is easy to ensure under standard assumptions about cryptographic hash functions. Such unpredictability helps security in a broad range of cases. For example, it makes it possible for applications to use a session ID from one connection to authenticate a session ID from another, thereby tying the two connections together. It furthermore helps ensure that TEPs do not trivially subvert the 33-byte minimum-length requirement for session IDs by padding shorter session IDs with zeros.

9. Experiments

This document has experimental status because TCP-ENO's viability depends on middlebox behavior that can only be determined a posteriori. Specifically, we need to determine to what extent middleboxes will permit the use of TCP-ENO. Once TCP-ENO is deployed, we will be in a better position to gather data on two types of failure:

1. Middleboxes downgrading TCP-ENO connections to unencrypted TCP. This can happen if middleboxes strip unknown TCP options or if they terminate TCP connections and relay data back and forth.

2. Middleboxes causing TCP-ENO connections to fail completely. This can happen if middleboxes perform deep packet inspection and start dropping segments that unexpectedly contain ciphertext, or

if middleboxes strip ENO options from non-SYN segments after allowing them in SYN segments.

Type-1 failures are tolerable since TCP-ENO is designed for incremental deployment anyway. Type-2 failures are more problematic, and, if prevalent, will require the development of techniques to avoid and recover from such failures. The experiment will succeed so long as we can avoid type-2 failures and find sufficient use cases that avoid type-1 failures (possibly along with a gradual path for further reducing type-1 failures).

In addition to the question of basic viability, deploying TCP-ENO will allow us to identify and address other potential corner cases or relaxations. For example, does the slight decrease in effective TCP segment payload pose a problem to any applications, which would require restrictions on how TEPs interpret socket buffer sizes? Conversely, can we relax the prohibition on default TEPs that disable urgent data?

A final important metric, related to the pace of deployment and incidence of type-1 failures, will be the extent to which applications adopt enhancements specific to TCP-ENO for endpoint authentication.

10. Security Considerations

An obvious use case for TCP-ENO is opportunistic encryption, e.g., encrypting some connections, but only where supported and without any kind of endpoint authentication. Opportunistic encryption provides a property known as "opportunistic security" [RFC7435], which protects against undetectable large-scale eavesdropping. However, it does not protect against detectable large-scale eavesdropping (for instance, if ISPs terminate TCP connections and proxy them or simply downgrade connections to unencrypted). Moreover, opportunistic encryption emphatically does not protect against targeted attacks that employ trivial spoofing to redirect a specific high-value connection to a man-in-the-middle attacker. Hence, the mere presence of TEP-indicated encryption does not suffice for an application to represent a connection as secure to the user.

Achieving stronger security with TCP-ENO requires verifying session IDs. Any application relying on ENO for communication security MUST incorporate session IDs into its endpoint authentication. By way of example, an authentication mechanism based on keyed digests (such as Digest Access Authentication [RFC7616]) can be extended to include the role and session ID in the input of the keyed digest. Authentication mechanisms with a notion of channel binding (such as Salted Challenge Response Authentication Mechanism (SCRAM) [RFC5802])

can be updated to derive a channel binding from the session ID. Higher-layer protocols MAY use the application-aware "a" bit to negotiate the inclusion of session IDs in authentication even when there is no in-band way to carry out such a negotiation. Because there is only one "a" bit, however, a protocol extension that specifies use of the "a" bit will likely require a built-in versioning or negotiation mechanism to accommodate crypto agility and future updates.

Because TCP-ENO enables multiple different TEPs to coexist, security could potentially be only as strong as the weakest available TEP. In particular, if TEPs use a weak hash function to incorporate the TCP-ENO transcript into session IDs, then an attacker can undetectably tamper with ENO options to force negotiation of a deprecated and vulnerable TEP. To avoid such problems, security reviewers of new TEPs SHOULD pay particular attention to the collision resistance of hash functions used for session IDs (including the state of cryptanalysis and research into possible attacks). Even if other parts of a TEP rely on more esoteric cryptography that turns out to be vulnerable, it ought nonetheless to be intractable for an attacker to induce identical session IDs at both ends after tampering with ENO contents in SYN segments.

Implementations MUST NOT send ENO options unless they have access to an adequate source of randomness [RFC4086]. Without secret unpredictable data at both ends of a connection, it is impossible for TEPs to achieve confidentiality and forward secrecy. Because systems typically have very little entropy on bootup, implementations might need to disable TCP-ENO until after system initialization.

With a regular three-way handshake (meaning no simultaneous open), the non-SYN-form ENO option in an active opener’s first ACK segment MAY contain N > 0 bytes of TEP-specific data, as shown in Figure 3. Such data is not part of the TCP-ENO negotiation transcript and therefore MUST be separately authenticated by the TEP.

11. IANA Considerations

This document defines a new TCP option kind for TCP-ENO, assigned a value of 69 from the TCP option space. This value is defined as:

Kind	Length	Meaning	Reference
69	N	Encryption Negotiation (TCP-ENO)	RFC 8547

Table 2: TCP Option Kind Numbers

Early implementations of TCP-ENO and a predecessor TCP encryption protocol made unauthorized use of TCP option kind 69. These earlier uses of option 69 are not compatible with TCP-ENO and could disable encryption or suffer complete connection failure when interoperating with TCP-ENO-compliant hosts. Hence, legacy use of option 69 MUST be disabled on hosts that cannot be upgraded to TCP-ENO. More recent implementations used experimental option 253 per [RFC6994] with 16-bit ExID 0x454E. Current and new implementations of TCP-ENO MUST use option 69, while any legacy implementations MUST migrate to option 69. Note in particular that Section 4.1 requires at most one SYN-form ENO option per segment, which means hosts MUST NOT include both option 69 and option 253 with ExID 0x454E in the same TCP segment.

This document defines a 7-bit glt field in the range of 0x20-0x7f. IANA has created and will maintain a new registry titled "TCP Encryption Protocol Identifiers" under the "Transmission Control Protocol (TCP) Parameters" registry. Table 3 shows the initial contents of this registry. This document allocates one TEP identifier (0x20) for experimental use. In case the TEP identifier space proves too small, identifiers in the range 0x70-0x7f are reserved to enable a future update to this document to define extended identifier values. Future assignments are to be made upon satisfying either of two policies defined in [RFC8126]: "IETF Review" or (for non-IETF stream specifications) "Expert Review with RFC Required". IANA will furthermore provide early allocation [RFC7120] to facilitate testing before RFCs are finalized.

Value	Meaning	Reference
0x20	Experimental Use	RFC 8547
0x70-0x7f	Reserved for extended values	RFC 8547

Table 3: TCP Encryption Protocol Identifiers

12. References

12.1. Normative References

[NIST-SP-800-57]
National Institute of Standards and Technology, "Recommendation for Key Management - Part 1: General", NIST Special Publication, 800-57, Revision 4, DOI 10.6028/NIST.SP.800-57pt1r4, January 2016, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf>.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Informative References

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, DOI 10.17487/RFC3493, February 2003, <https://www.rfc-editor.org/info/rfc3493>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <https://www.rfc-editor.org/info/rfc5382>.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, DOI 10.17487/RFC5802, July 2010, <https://www.rfc-editor.org/info/rfc5802>.

[RFC6394] Barnes, R., "Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)", RFC 6394, DOI 10.17487/RFC6394, October 2011, <https://www.rfc-editor.org/info/rfc6394>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <https://www.rfc-editor.org/info/rfc7435>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <https://www.rfc-editor.org/info/rfc7616>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Acknowledgments

We are grateful for contributions, help, discussions, and feedback from the IETF and its TCPINC Working Group, including Marcelo Bagnulo, David Black, Bob Briscoe, Benoit Claise, Spencer Dawkins, Jake Holland, Jana Iyengar, Tero Kivinen, Mirja Kuhlewind, Watson Ladd, Kathleen Moriarty, Yoav Nir, Christoph Paasch, Eric Rescorla, Adam Roach, Kyle Rose, Michael Scharf, Joe Touch, and Eric Vyncke. This work was partially funded by DARPA CRASH and the Stanford Secure Internet of Things Project.

Contributors

Dan Boneh was a coauthor of the draft that became this document.

Authors' Addresses

Andrea Bittau
Google
345 Spear Street
San Francisco, CA 94105
United States of America

Email: bittau@google.com

Daniel B. Giffin
Stanford University
353 Serra Mall, Room 288
Stanford, CA 94305
United States of America

Email: daniel@beech-grove.net

Mark Handley
University College London
Gower St.
London WC1E 6BT
United Kingdom

Email: M.Handley@cs.ucl.ac.uk

David Mazieres
Stanford University
353 Serra Mall, Room 290
Stanford, CA 94305
United States of America

Email: dm@uun.org

Eric W. Smith
Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304
United States of America

Email: eric.smith@kestrel.edu