Network Working Group
Internet-Draft
Intended status: Experimental
Expires: March 13, 2016

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

September 10, 2015

# TCP-ENO: Encryption Negotiation Option
## draft-bittau-tcpinc-tcpeno-02

Abstract

Despite growing adoption of TLS [RFC5246], 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, making incremental deployment impossible. Second, legacy applications themselves cannot always be upgraded, requiring 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.

Table of Contents



## 1. Requirements language

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 [RFC2119].

## 2. 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.

While the need for encryption is immediate, future developments could alter trade-offs and change the best approach to TCP-level encryption (beyond introducing new cipher suites). For example:

- Increased option space in TCP [I-D.ietf-tcpm-tcp-edo][I-D.briscoe-tcpm-inspace-mode-tcpbis][I-D.touch-tcpm-tcp-syn-ext-opt] could reduce round trip times and simplify protocols.

- API revisions to socket interfaces [RFC3493] could benefit from integration with TCP-level encryption, particularly if combined with technologies such as DANE [RFC6394].

- The forthcoming TLS 1.3 [I-D.ietf-tls-tls13] standard could reach more applications given an out-of-band, backward-compatible mechanism for enabling encryption.

- TCP fast open [RFC7413], as it gains more widespread adoption and middlebox acceptance, could potentially benefit from tailored encryption support.

- Cryptographic developments that either shorten or lengthen the minimal key exchange messages required could affect how such messages are best encoded in TCP segments.

Introducing TCP options, extending operating system interfaces to support TCP-level encryption, and extending applications to take advantage of TCP-level encryption will all require effort. To the greatest extent possible, this effort ought to remain applicable if the need arises 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, and

2. How to perform encryption at the TCP layer.

This document addresses question 1 with a new option called TCP-ENO. TCP-ENO provides a framework in which two endpoints can agree on one among multiple possible TCP encryption _specs_. For future

compatibility, encryption specs can vary widely in terms of wire format, use of TCP option space, and integration with the TCP header and segmentation. A companion document, the TCPINC encryption spec, addresses question 2. TCPINC enables TCP-level traffic encryption today. TCP-ENO ensures that the effort invested to deploy TCPINC can benefit future encryption specs should a different approach at some point be preferable.

At a lower level, TCP-ENO was designed to achieve the following goals:

1. Enable endpoints to negotiate the use of a separately specified encryption _spec_.

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

3. Provide 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. Provide a standard negotiation transcript through which specs can 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.

## 3. The TCP-ENO option

TCP-ENO is a TCP option used during connection establishment to negotiate how to encrypt traffic. As an option, TCP-ENO can be deployed incrementally. Legacy hosts unaware of the option simply ignore it and never send it, causing traffic to fall back to unencrypted TCP. Similarly, middleboxes that strip out unknown options including TCP-ENO will downgrade connections to plaintext without breaking them. Of course, downgrading makes TCP-ENO vulnerable to active attackers, but appropriately modified applications can protect themselves by considering the state of TCP-level encryption during authentication, as discussed in Section 7.

The ENO option takes two forms. In TCP segments with the SYN flag set, it acts as a container for a series of one or more suboptions, labeled "Opt_0", "Opt_1", ... in Figure 1. In non-SYN segments, ENO conveys only a single bit of information, namely an acknowledgment

that the sender received an ENO option in the other host's SYN segment. (Such acknowledgments enable graceful fallback to unencrypted TCP in the event that a middlebox strips ENO options in one direction.) Figure 2 illustrates the non-SYN form of the ENO option. Encryption specs MAY include extra bytes in a non-SYN ENO option, but TCP-ENO itself MUST ignore them. In accordance with TCP [RFC0793], the first two bytes of the ENO option always consist of the kind (ENO) and the total length of the option.

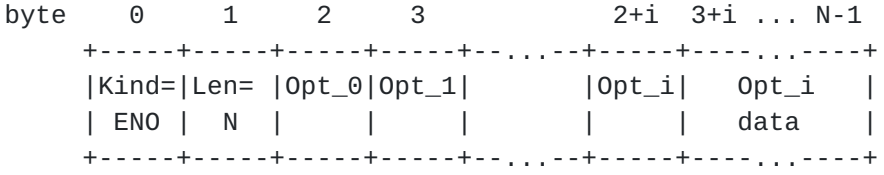



Figure 1: TCP-ENO option in SYN segment (MUST contain at least one suboption)

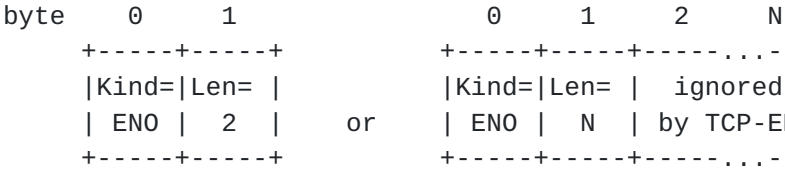
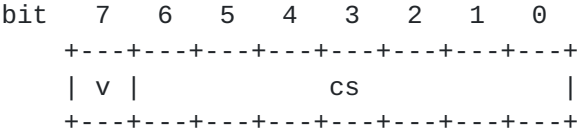



Figure 2: non-SYN TCP-ENO option in segment without SYN flag

Every suboption starts with a byte of the form illustrated in Figure 3. The seven-bit value "cs" specifies the meaning of the suboption. Each value of "cs" either specifies general parameters (discussed in Section 3.3) or indicates the willingness to use a specific encryption spec detailed in a separate document.




v - 1 when suboption followed by variable-length data
cs - global configuration option or encryption spec identifier

Figure 3: Format of suboption byte

The high bit "v" in a suboption's first byte specifies whether or not the suboption is followed by variable-length data. If "v" is 0, the suboption consists of only the one byte shown in Figure 3. If "v" is 1, then the suboption is followed by variable-length data. Suboption data MAY be used for session caching, cipher suite negotiation, key exchange, or other purposes, as determined by the value of "cs".

Every suboption but the last in an ENO option MUST be a one-byte suboption (with "v" = 0). The last suboption MAY be a variable-length suboption. Its length is determined by the total length of the TCP option. In Figure 1, "Opt_i" is the variable-length option; its total size is N-(2+i) bytes--one byte for "Opt_i" itself and N-(3+i) bytes for additional data. Multiple suboptions with data may be included in a single TCP SYN segment by repeating the ENO option.

Table 1 summarizes the allocation of values of "cs". Values under 0x10 are assigned to _general suboptions_ whose meaning applies across encryption specs, as discussed in Section 3.3. Values greater than or equal to 0x20 will be assigned to _spec identifiers_. Values in the range 0x10-0x1f are reserved for possible future general options. Implementations MUST ignore all unknown suboptions.

| cs | Meaning |
|---|---|
| 0x00-0x0f | General options (see Section 3.3) |
| 0x10-0x1f | Reserved for possible use by future general options |
| 0x20-0x7f | Used to designate encryption specs |

Table 1: Allocation of cs bits in TCP-ENO suboptions

## 3.1. TCP-ENO roles

TCP-ENO uses abstract roles to distinguish the two ends of a TCP connection: One host plays the "A" role, while the other host plays the "B" role. Following a normal three-way handshake, the active opener plays the A role and the passive opener plays the B role. An active opener is a host that sends a SYN segment without the ACK flag set (after a "connect" system call on socket-based systems). A passive opener's SYN segment always contains the ACK flag (and follows a "listen" call on socket-based systems).

Roles are abstracted from the active/passive opener distinction to deal with simultaneous open, in which both hosts are active openers. For simultaneous open, the general suboptions discussed in Section 3.3 define a tie-breaker bit "b", where the host with "b = 1" plays the B role, and the host with "b = 0" plays the A role. If two active openers have the same "b" bit, TCP-ENO fails and reverts to unencrypted TCP.

More precisely, the above role assignment can be reduced to comparing a two-bit role _priority_ for each host, shown in Figure 4. The most significant bit, "p", is 1 for a passive opener and 0 for an active opener. The least-significant bit "b" is the tie-breaker bit. The

host with the lower priority assumes the A role; the host with the higher priority assumes the B role. In the event of a tie, TCP-ENO fails and MUST continue with unencrypted TCP as if the ENO options had not been present in SYN segments.

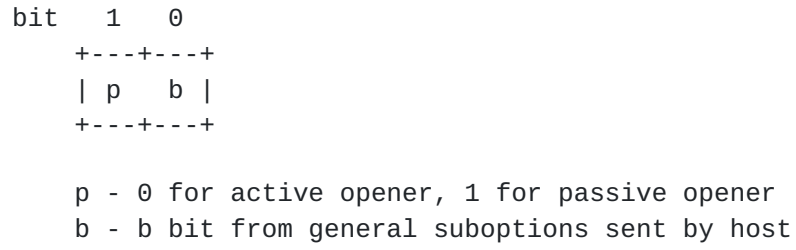



Figure 4: Role priority of an endpoint

Encryption specs 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 role A and B respectively in a connection.

## 3.2. TCP-ENO handshake

The TCP-ENO option is intended for use during TCP connection establishment. To enable incremental deployment, a host needs to ensure both that the other host supports TCP-ENO and that no middlebox has stripped the ENO option from its own TCP segments. In the event that either of these conditions does not hold, implementations MUST immediately cease sending TCP-ENO options and MUST continue with unencrypted TCP as if the ENO option had not been present.

More precisely, for negotiation to succeed, the TCP-ENO option MUST be present in the SYN segment sent by each host, so as to indicate support for TCP-ENO. Additionally, the ENO option MUST be present in the first ACK segment sent by each host, so as to indicate that no middlebox stripped the ENO option from the ACKed SYN. Depending on whether a host is an active or a passive opener, the first ACK segment may or may not be the same as the SYN segment. Specifically:

- An active opener begins with a SYN-only segment, and hence must send two segments containing ENO options. The initial SYN-only segment MUST contain an ENO option with at least one suboption, as pictured in Figure 1. If ENO succeeds, the active opener's first ACK segment MUST subsequently contain a non-SYN ENO option, as pictured in Figure 2.

- A passive opener's first transmitted segment has both the SYN and ACK flags set. Therefore, a passive opener sends an ENO option of

the type shown in Figure 1 in its single SYN-ACK segment and does not send a non-SYN ENO option.

A spec identifier in one host's SYN segment is _valid_ if it is compatible with a suboption in the other host's SYN segment. Two suboptions are _compatible_ when they have the same "cs" value (>= 0x20) and when the particular combination of "v" bits and suboption data in suboptions of the two SYN segments is well-defined by the corresponding encryption spec. Specs MAY allow or disallow any combination of values of "v" in the two SYN segments.

Once the two sides have exchanged SYN segments, the _negotiated spec_ is the last valid spec identifier in the SYN segment of host B (that is, the passive opener in the absence of simultaneous open). In other words, the order of suboptions in host B's SYN segment determines spec priority, while the order of suboptions in host A's SYN segment has no effect. Hosts must disable TCP-ENO if there is no valid spec in host B's SYN segment. Note that negotiation prioritizes the last rather than the first valid suboption so as to favor the spec with suboption data, if there is one.

When possible, host B SHOULD send only one spec identifier (suboption in the range 0x20-0xff), and SHOULD ensure this option is valid. However, sending a single valid spec identifier is not required, as doing so could be impractical in some cases, such as simultaneous open or library-level implementations that can only provide a static TCP-ENO option to the kernel.

A host MUST disable ENO if any of the following conditions holds:

1. The host receives a SYN segment without an ENO option,

2. The host receives a SYN segment that contains no valid encryption specs when paired with the SYN segment that the host has already sent or would otherwise have sent,

3. The host receives a SYN segment containing general suboptions that are incompatible with the SYN segment that it has already sent or would otherwise have sent, or

4. The first ACK segment received by a host does not contain an ENO option.

After disabling ENO, a host MUST NOT transmit any further ENO options and MUST fall back to unencrypted TCP.

Conversely, if a host receives an ACK segment containing an ENO option, then encryption MUST be enabled. From this point the host

MUST follow the encryption protocol of the negotiated spec and MUST NOT present raw TCP payload data to the application. In particular, data segments MUST contain ciphertext or key agreement messages as determined by the negotiated spec, and MUST NOT contain plaintext application data.

### 3.2.1. Handshake examples

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

Figure 5: Three-way handshake with successful TCP-ENO negotiation

Figure 5 shows a three-way handshake with a successful TCP-ENO negotiation. The two sides agree to follow the encryption spec 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 6: Three-way handshake with failed TCP-ENO negotiation

Figure 6 shows a failed TCP-ENO negotiation. The active opener (A) indicates support for specs corresponding to suboptions X and Y. Unfortunately, at this point one of thee things occurs:

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

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

3. 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<X>    [ENO stripped by middlebox]
(3) A -> B:  ACK
[rest of connection unencrypted legacy TCP]
```

Figure 7: Failed TCP-ENO negotiation because of network filtering

Figure 7 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. Hence, A does not receive an ENO option from B, disables ENO, and does not include the required non-SYN ENO option when ACKing the other host's SYN segment. The lack of ENO in A's ACK segment signals to B that the connection will not be encrypted. At this point, the two hosts proceed with an unencrypted TCP connection.

```
(1) A -> B:  SYN      ENO<Y,X>
(2) B -> A:  SYN      ENO<0x01,X,Y,Z>
(3) A -> B:  SYN-ACK  ENO<Y,X>
(4) B -> A:  SYN-ACK  ENO<0x01,X,Y,Z>
[rest of connection encrypted according to spec for Y]
```

Figure 8: Simultaneous open with successful TCP-ENO negotiation

Figure 8 shows a successful TCP-ENO negotiation with simultaneous open. Here the first four segments MUST contain an ENO option, as each side sends both a SYN-only and a SYN-ACK segment. The ENO option in each hosts'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. Note the use of the tie-breaker bit in general suboption 0x01 assigns B its role, as discussed in Section 3.3. The last valid spec in B's ENO option is Y, so Y is the negotiated spec.

## 3.3. General suboptions

Suboptions 0x00-0x0f are used for general conditions that apply regardless of the negotiated encryption spec. A TCP segment MUST include at most one ENO suboption whose high nibble is 0. The value of the low nibble is interpreted as a bitmask, illustrated in Figure 9.

```
bit   7   6   5   4   3   2   1   0
    +---+---+---+---+---+-------+---+
    | 0   0   0   0   z    aa     b |
    +---+---+---+---+---+-------+---+

    z  - Zero bit (reserved for future use)
    aa - Application-aware bits
    b  - Tie-breaker bit for simultaneous open
```

Figure 9: Format of the general option byte

The fields of the bitmask are interpreted as follows:

z The "z" bit is reserved for future revisions of TCP-ENO. Its value MUST be set to zero in sent segments and ignored in received segments.

aa The two application-aware bits indicate that the application on the sending host is aware of TCP-ENO and has been extended to alter its behavior in the presence of encrypted TCP. There are four possible values, as shown in Table 2. The default, when applications have not been modified to take advantage of TCP-ENO, MUST be 00. However, implementations SHOULD provide an API through which applications can set the bits to other values and query for the other host's application-aware bits. The value 01 indicates that the application is aware of TCP-ENO. The value 10 (binary) is reserved for future use. It MUST be interpreted as the application being aware of TCP-ENO, but MUST never be sent.

Value 11 (binary) indicates that an application is aware of TCP-ENO and requires application awareness from the other side. If one host sends value 00 and the other host sends 11, then TCP-ENO MUST be disabled and fall back to unencrypted TCP. Any other combination of values (including the reserved 10) is compatible with enabling encryption. A possible use of value 11 is for applications that perform legacy encryption and wish to disable TCP-ENO unless higher-layer encryption can be disabled.

| Value | Meaning |
|---|---|
| 00 | Application is not aware of TCP-ENO |
| 01 | Application is aware of TCP-ENO |
| 10 | Reserved but interpreted as ENO-aware |
| 11 | Application awareness is mandatory for use of TCP-ENO |

Table 2: Meaning of the two application-aware bits

b This is the tie-breaker bit in role priority, discussed in Section 3.1.

A SYN segment without an explicit general suboption has an implicit general suboption of 0x00.

## 3.4. Negotiation transcript

To defend against attacks on encryption negotiation itself, encryption specs need a way to reference a transcript of TCP-ENO's negotiation. In particular, an encryption spec MUST fail with high

probability if its selection resulted from tampering with or forging initial SYN segments.

TCP-ENO defines its negotiation transcript as a packed data structure consisting of a series of TCP-ENO options (each including the ENO and length bytes, as they appeared in the TCP header). Specifically, the transcript is constructed from the following, in order:

1. Every TCP-ENO option in host A's SYN segment, including the kind and length bytes, in the order the options appeared in that SYN segment.

2. A minimal two-byte ENO option, as shown on the left in Figure 2.

3. Every TCP-ENO option in host B's SYN segment, including the kind and length bytes, in the order the options appeared in that SYN segment.

4. A minimal two-byte ENO option, as shown on the left in Figure 2.

Note that 2 and 4 merely serve as delimiters to separate the two hosts' options from each other and from any data that follows the transcript. Note further that any ignored data in non-SYN ENO options does not appear in the transcript. Because parts 2 and 4 are always exactly two bytes and SYN segments MUST NOT contain two-byte ENO options, this encoding is unambiguous.

For the transcript to be well defined, hosts MUST NOT alter ENO options in retransmitted segments, or between the SYN and SYN-ACK segments of a simultaneous open, except that an active opener MAY remove the ENO option altogether from a retransmitted SYN-only segment and disable TCP-ENO. Such removal could be useful if middleboxes are dropping segments with the ENO option.

## 4. Requirements for encryption specs

TCP-ENO was designed to afford encryption spec authors a large amount of design flexibility. Nonetheless, to fit all encryption specs into a coherent framework and abstract most of the differences away for application writers, all encryption specs claiming ENO "cs" numbers MUST satisfy the following properties.

- Specs MUST protect TCP data streams with authenticated encryption.

- Specs 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 spec. Section 4.1 describes the requirements of the session ID in more detail.

- Specs MUST NOT permit the negotiation of any encryption algorithms with significantly less than 128-bit security.

- Specs 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.)

- Specs MUST NOT allow the negotiation of encryption modes that do not provide forward secrecy some bounded, short time after the close of a TCP connection.

- Specs MUST protect and authenticate the end-of-file marker traditionally conveyed by TCP's FIN flag when the remote application calls "close" or "shutdown". However, end-of-file MAY be conveyed though a mechanism other than TCP FIN. Moreover, specs MAY permit attacks that cause TCP connections to abort, but such an abort MUST raise an error that is distinct from an end-of-file condition.

- Specs MAY disallow the use of TCP urgent data by applications, but MUST NOT allow attackers to manipulate the URG flag and urgent pointer in ways that are visible to applications.

## 4.1. Session IDs

Each spec MUST define a session ID that uniquely identifies each encrypted TCP connection. Implementations SHOULD expose the session ID to applications via an API extension. Applications that are aware of TCP-ENO SHOULD incorporate the session ID value and TCP-ENO role (A or B) into any authentication mechanisms layered over TCP encryption so as to authenticate actual TCP endpoints.

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 deviating from the encryption spec, to establish two TCP connections with the same session ID to remote hosts obeying the spec.

To prevent session IDs from being confused across specs, all session IDs begin with the negotiated spec identifier--that is, the last valid spec identifier in host B's SYN segment. If the "v" bit was 1 in host B's SYN segment, then it is also 1 in the session ID. However, only the first byte is included, not the suboption data. Figure 10 shows the resulting format. This format is designed for spec authors to compute unique identifiers; it is not intended for

applications 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.

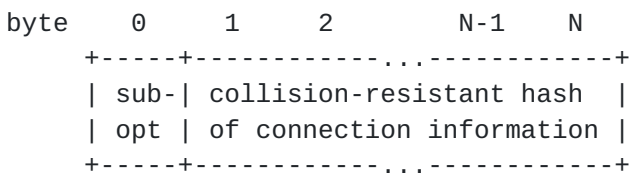



Figure 10: Format of a session ID

Though specs retain considerable flexibility in their definitions of the session ID, all session IDs MUST meet certain minimum requirements. In particular:

- The session ID MUST be at least 33 bytes (including the one-byte suboption), though specs may choose longer session IDs.
- The session ID MUST depend in a collision-resistant way on fresh data contributed by both sides of the connection.
- The session ID MUST depend in a collision-resistant way on any public keys, public Diffie-Hellman parameters, or other public asymmetric cryptographic parameters that are employed by the encryption spec and have corresponding private data that is known by only one side of the connection.
- 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 chose to make session IDs public. Therefore, specs MUST NOT place any confidential data in the session ID (such as data permitting the derivation of session keys).
- The session ID MUST depend on the negotiation transcript specified in Section 3.4 in a collision-resistant way.

## 4.2. Option kind sharing

This draft does not specify the use of ENO options in any segments other than the initial SYN and ACK segments of a connection. Moreover, it does not specify the content of ENO options in an initial ACK segment that has the SYN flag clear. As a result, any use of the ENO option kind after the SYN exchange will not conflict with TCP-ENO. Therefore, encryption specs that require TCP option

space MAY re-purpose the ENO option kind for use in segments after the initial SYN.

## 5. API extensions

Implementations SHOULD provide API extensions through which applications can query and configure the behavior of TCP-ENO, including retrieving session IDs, setting and reading application-aware bits, and specifying which specs to negotiate. The specifics of such an API are outside the scope of this document.

## 6. Open issues

This document has experimental status because of several open issues. Some questions about TCP-ENO's viability depend on middlebox behavior that can only be determined a posteriori. Hence, initial deployment of ENO will be an experiment. In addition, a few design questions exists on which consensus is not clear, and hence for which greater discussion and justification of TCP-ENO's design may be helpful.

### 6.1. Experiments

One of the primary open questions is 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 applications perform deep packet inspection and start dropping segments that unexpectedly contain ciphertext.

The first type of failure is tolerable since TCP-ENO is designed for incremental deployment anyway. The second type of failure is more problematic, and, if prevalent, will require the development of techniques to avoid and recover from such failures.

### 6.2. Simultaneous open

Simultaneous open is the only way to establish a TCP connection between TCP hosts in certain NAT configurations [RFC5382]. The principle challenge in simultaneous open is breaking TCP's symmetry for both sides to agree on the assignment of the A and B roles. Relying on TCP/IP header fields such as the IP address, port number, and initial sequence number is problematic as these values may be

modified by middleboxes, meaning a sender does not know what values the recipient will see for these fields.

The authors lack data on how prevalent simultaneous open is in the wild. The use of simultaneous open has been specified for ICE [RFC6544], but the highest profile implementation (the firefox browser) currently prefers UDP over TCP when permitted by firewalls. Moreover, applications of ICE typically already encrypt data and would disable TCP-ENO to avoid double encryption. It is therefore unclear what level of support TCP-ENO should provide for simultaneous open, or at what cost such support is justified. The working group has discussed four levels of support with no clear consensus:

1. Require applications to break the tie out of band and assign themselves A and B roles. If applications do not assign the roles properly, the TCP connection fails entirely.

2. As above, require applications to specify roles, but if they do so incorrectly fall back to unencrypted TCP.

3. Require applications to declare that they are using simultaneous open, but do not require them to negotiate roles. Leave it to TCP-ENO break the tie and negotiate roles.

4. Design TCP-ENO so that it works completely transparently in conjunction with simultaneous open, with no application involvement required.

This simplest and cheapest solution is obviously #1. This document currently embraces design point #2, at the cost of an extra bit (the "b" bit in the general suboption) for hosts to check whether roles were properly assigned. Solution #3 would likely consume 4-8 additional bytes of option space in the case of a simultaneous open, so as to include a random tie-breaker value. Solution #4 would consume 4-8 additional bytes of option space in every SYN segment, as current APIs make it impossible to distinguish a "connect" call intended for a simultaneous open from one intended for a three-way handshake.

### 6.3. Multiple Session IDs

Though currently specs must output a single session ID, it might alternatively be useful to define multiple identifiers per connection. As an example, a public session ID might be used to authenticate a connection, while a private session ID could be used as an authentication key to link out-of-band data (such as another TCP connection) to the original connection.

### 6.4. Suboption data

TCP-ENO currently optimizes for the case that a single suboption per SYN segment contains suboption data. This design was chosen in expectation that the following two use cases will be the most common:

- An active opener advertises support for multiple specs using one-byte suboptions. The passive opener picks one of the advertised specs and replies with a single suboption, possibly using suboption data for options within the negotiated spec. Such spec-specific options might convey supported elliptic curves or public key ciphers.

- An active opener advertises support for multiple specs as above, but also includes a single longer suboption containing a session caching cookie with which the hosts may be able to avoid the cost of public key cryptography. In this case, the server either accepts the cookie or reverts to picking one of the other specs as in the previous case.

Both of these use cases require at most one multi-byte suboption per SYN segment. To optimize for this case, TCP-ENO relies on the TCP option length byte to specify the length of the multi-byte suboption implicitly. Segments with more than one multi-byte suboption must repeat the ENO kind byte, losing one byte of precious TCP SYN option space.

An alternative would be for each multi-byte suboption to be followed by its own length field. This would cost an extra byte of SYN option space in the two cases above, but save one byte for each additional multi-byte suboption.

As an example, in the current ENO design, a SYN segment with ENO suboption containing 2 bytes of data consumes 5 bytes (the ENO kind, the TCP option length, the spec identifier, and 2 bytes of suboption data). An ENO option with two 2-byte suboptions requires double this, or 10 bytes. By contrast, in a design with a suboption length byte, one 2-byte suboption would cost 6 bytes (ENO kind, TCP option length, suboption, suboption length, and 2 bytes of option data), but two 2-byte suboptions could be packed together, without repeating the ENO kind byte, in only 9 bytes of option space.

In the event that the above two use cases are not the most prevalent, it may be worth revisiting ENO's choice of optimized case.

## 7. Security considerations

An obvious use case for TCP-ENO is opportunistic encryption. However, if applications do not check and verify the session ID, they will be open to man-in-the-middle attacks as well as simple downgrade attacks in which an attacker strips off the TCP-ENO option. Hence, where possible, applications SHOULD be modified to fold the session ID into authentication mechanisms, and SHOULD employ the application-aware bits as needed to enable such negotiation in a backward-compatible way.

Because TCP-ENO enables multiple different encryption specs to coexist, security could potentially be only as strong as the weakest available encryption spec. For this reason, it is crucial for session IDs to depend on the TCP-ENO transcript in a strong way. Hence, encryption specs SHOULD compute session IDs using only well-studied and conservative hash functions. Thus, even if an encryption spec is broken, and even if people deprecate it instead of disabling it, and even if an attacker tampers with ENO options to force negotiation of the broken spec, it should still be intractable for the attacker to induce identical session IDs at both hosts.

Implementations MUST not send ENO options unless encryption specs have access to a strong source of randomness or pseudo-randomness. Without secret unpredictable data at both ends of a connection, it is impossible for encryption specs to satisfy the confidentiality and forward secrecy properties required by this document.

## 8. IANA Considerations

A new TCP option kind number needs to be assigned to ENO by IANA.

In addition, IANA will need to maintain an ENO suboption registry mapping suboption "cs" values to encryption specs.

## 9. Acknowledgments

This work was funded by DARPA CRASH under contract #N66001-10-2-4088.

## 10. References

### 10.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://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, <http://www.rfc-editor.org/info/rfc2119>.

## 10.2. Informative References

[I-D.briscoe-tcpm-inspace-mode-tcpbis]
Briscoe, B., "Inner Space for all TCP Options (Kitchen Sink Draft - to be Split Up)", draft-briscoe-tcpm-inspace-mode-tcpbis-00 (work in progress), March 2015.

[I-D.ietf-tcpm-tcp-edo]
Touch, J. and W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo-03 (work in progress), April 2015.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-07 (work in progress), July 2015.

[I-D.touch-tcpm-tcp-syn-ext-opt]
Touch, J. and T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt-02 (work in progress), April 2015.

[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, <http://www.rfc-editor.org/info/rfc3493>.

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

[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, <http://www.rfc-editor.org/info/rfc5382>.

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

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <http://www.rfc-editor.org/info/rfc6544>.

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

## Authors' Addresses

Andrea Bittau
Stanford University
353 Serra Mall, Room 288
Stanford, CA 94305
US

Email: bittau@cs.stanford.edu

Dan Boneh
Stanford University
353 Serra Mall, Room 475
Stanford, CA 94305
US

Email: dabo@cs.stanford.edu

Daniel B. Giffin
Stanford University
353 Serra Mall, Room 288
Stanford, CA 94305
US

Email: dbg@scs.stanford.edu

Mark Handley
University College London
Gower St.
London WC1E 6BT
UK

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

David Mazieres
Stanford University
353 Serra Mall, Room 290
Stanford, CA 94305
US

Email: dm@uun.org

Eric W. Smith
Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304
US

Email: eric.smith@kestrel.edu