Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

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

October 31, 2016

Cryptographic protection of TCP Streams (tcpcrypt)
draft-ietf-tcpinc-tcpcrypt-03

Abstract

This document specifies tcpcrypt, a TCP encryption protocol designed for use in conjunction with the TCP Encryption Negotiation Option (TCP-ENO) [I-D.ietf-tcpinc-tcpeno]. Tcpcrypt coexists with middleboxes by tolerating resegmentation, NATs, and other manipulations of the TCP header. The protocol is self-contained and specifically tailored to TCP implementations, which often reside in kernels or other environments in which large external software dependencies can be undesirable. Because the size of TCP options is limited, the protocol requires one additional one-way message latency to perform key exchange before application data may be transmitted. However, this cost can be avoided between two hosts that have recently established a previous tcpcrypt connection.

This Internet-Draft will expire on May 4, 2017.

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

This document describes tcpcrypt, an extension to TCP for cryptographic protection of session data. Tcpcrypt was designed to meet the following goals:

- Meet the requirements of the TCP Encryption Negotiation Option (TCP-ENO) [I-D.ietf-tcpinc-tcpeno] for protecting connection data.
- Be amenable to small, self-contained implementations inside TCP stacks.
- Minimize additional latency at connection startup.
- As much as possible, prevent connection failure in the presence of NATs and other middleboxes that might normalize traffic or otherwise manipulate TCP segments.
- Operate independently of IP addresses, making it possible to authenticate resumed sessions efficiently even when either end changes IP address.

3. Encryption protocol

This section describes the tcpcrypt protocol at an abstract level. The concrete format of all messages is specified in Section 4.

3.1. Cryptographic algorithms

Setting up a tcpcrypt connection employs three types of cryptographic algorithms:

- A _key agreement scheme_ is used with a short-lived public key to agree upon a shared secret.
- An _extract function_ is used to generate a pseudo-random key from some initial keying material, typically the output of the key agreement scheme. The notation Extract(S, IKM) denotes the output of the extract function with salt S and initial keying material IKM.
- A _collision-resistant pseudo-random function (CPRF)_ is used to generate multiple cryptographic keys from a pseudo-random key, typically the output of the extract function. We use the notation CPRF(K, CONST, L) to designate the output of L bytes of the pseudo-random function identified by key K on CONST.

The Extract and CPRF functions used by default are the Extract and Expand functions of HKDF [RFC5869]. These are defined as follows in terms of the PRF "HMAC-Hash(key, value)" for a negotiated "Hash" function:

```
HKDF-Extract(salt, IKM) -> PRK
   PRK = HMAC-Hash(salt, IKM)

HKDF-Expand(PRK, CONST, L) -> OKM
   T(0) = empty string (zero length)
   T(1) = HMAC-Hash(PRK, T(0) | CONST | 0x01)
   T(2) = HMAC-Hash(PRK, T(1) | CONST | 0x02)
   T(3) = HMAC-Hash(PRK, T(2) | CONST | 0x03)
   ...

   OKM  = first L octets of T(1) | T(2) | T(3) | ...
```

Figure 1: The symbol | denotes concatenation, and the counter concatenated to the right of CONST is a single octet.

Lastly, once tcpcrypt has been successfully set up, an _authenticated encryption mode_ is used to protect the confidentiality and integrity of all transmitted application data.

3.2. Protocol negotiation

Tcpcrypt depends on TCP-ENO [I-D.ietf-tcpinc-tcpeno] to negotiate whether encryption will be enabled for a connection, and also which key agreement scheme to use. TCP-ENO negotiates the use of a particular TCP encryption protocol or _TEP_ by including protocol identifiers in ENO suboptions. This document associates four TEP identifiers with the tcpcrypt protocol, as listed in Table 1. Future standards may associate additional identifiers with tcpcrypt.

An active opener that wishes to negotiate the use of tcpcrypt will include an ENO option in its SYN segment. That option will include suboptions with TEP identifiers indicating the key-agreement schemes it is willing to enable. The active opener MAY additionally include suboptions indicating support for encryption protocols other than tcpcrypt, as well as other general options as specified by TCP-ENO.

If a passive opener receives an ENO option including tcpcrypt TEPs it supports, it MAY then attach an ENO option to its SYN-ACK segment, including _solely_ the TEP it wishes to enable.

To establish distinct roles for the two hosts in each connection, tcpcrypt depends on the role-negotiation mechanism of TCP-ENO [I-D.ietf-tcpinc-tcpeno]. As part of the negotiation process, TCP-ENO assigns hosts unique roles abstractly called "A" at one end of the connection and "B" at the other. Generally, an active opener plays the "A" role and a passive opener plays the "B" role; but in the case of simultaneous open, an additional mechanism breaks the symmetry and assigns different roles to the two hosts. This document adopts the terms "host A" and "host B" to identify each end of a connection uniquely, following TCP-ENO’s designation.

Once two hosts have exchanged SYN segments, the _negotiated TEP_ is the last TEP identifier in the SYN segment of host B (that is, the passive opener in the absence of simultaneous open) that also occurs in that of host A. If there is no such TEP, hosts MUST disable TCP-ENO and tcpcrypt.

The _negotiated suboption_ is the ENO suboption from the SYN segment of host B that contains the negotiated TEP, if it exists. This suboption includes a one-bit flag "v" which indicates the presence of additional data. For tcpcrypt TEPs, if the negotiated suboption contains "v = 0", a fresh key agreement will be perfomed as described below in Section 3.3. If it contains "v = 1", it is a _resumption suboption_: this indicates that the key-exchange messages will be omitted in favor of determining keys via session-caching as described in Section 3.4, and protected application data may immediately be sent as detailed in Section 3.5.

Note that the negotiated TEP is determined without reference to the "v" bits in ENO suboptions, so if host A offers a resumption suboption with a particular TEP and host B replies with a non-resumption suboption with the same TEP, that may become the negotiated suboption and fresh key agreement will be performed. That is, sending a resumption suboption also implies willingness to perform fresh key-exchange with the indicated TEP.

As required by TCP-ENO, once a host has both sent and received an ACK segment containing an ENO option, encryption MUST be enabled and plaintext application data MUST NOT ever be exchanged on the connection. If the negotiated TEP is among those listed in Table 1, a host MUST follow the protocol described in this document.

3.3. Key exchange

Following successful negotiation of a tcpcrypt TEP, all further signaling is performed in the Data portion of TCP segments. Except when resumption was negotiated (described below in Section 3.4), the two hosts perform key exchange through two messages, "Init1" and "Init2", at the start of the data streams of host A and host B, respectively. These messages may span multiple TCP segments and need not end at a segment boundary. However, the segment containing the last byte of an "Init1" or "Init2" message SHOULD have TCP’s PSH bit set.

The key exchange protocol, in abstract, proceeds as follows:

```
A -> B:  Init1 = { INIT1_MAGIC, sym-cipher-list, N_A, PK_A }
B -> A:  Init2 = { INIT2_MAGIC, sym-cipher, N_B, PK_B }
```

The concrete format of these messages is specified in further detail in Section 4.1.

The parameters are defined as follows:

- "INIT1_MAGIC", "INIT2_MAGIC": constants defined in Table 3.
- "sym-cipher-list": a list of symmetric ciphers (AEAD algorithms) acceptable to host A. These are specified in Table 2.
- "sym-cipher": the symmetric cipher selected by host B from the "sym-cipher-list" sent by host A.
- "N_A", "N_B": nonces chosen at random by hosts A and B, respectively.

- "PK_A", "PK_B": ephemeral public keys for hosts A and B, respectively. These, as well as their corresponding private keys, are short-lived values that SHOULD be refreshed periodically. The private keys SHOULD NOT ever be written to persistent storage.

The ephemeral secret ("ES") is defined to be the result of the key-agreement algorithm whose inputs are the local host's ephemeral private key and the remote host's ephemeral public key. For example, host A would compute "ES" using its own private key (not transmitted) and host B's public key, "PK_B".

The two sides then compute a pseudo-random key ("PRK"), from which all session keys are derived, as follows:

```
PRK = Extract (N_A, eno-transcript | Init1 | Init2 | ES)
```

Above, "|" denotes concatenation; "eno-transcript" is the protocol-negotiation transcript defined in TCP-ENO; and "Init1" and "Init2" are the transmitted encodings of the messages described in Section 4.1.

A series of "session secrets" and corresponding session identifiers are then computed from "PRK" as follows:

```
ss[0] = PRK
ss[i] = CPRF (ss[i-1], CONST_NEXTK, K_LEN)

SID[i] = CPRF (ss[i], CONST_SESSID, K_LEN)
```

The value "ss[0]" is used to generate all key material for the current connection. "SID[0]" is the _bare session ID_ for the current connection, and will with overwhelming probability be unique for each individual TCP connection.

The values of "ss[i]" for "i > 0" can be used to avoid public key cryptography when establishing subsequent connections between the same two hosts, as described in Section 3.4. The "CONST_*" values are constants defined in Table 3. The length "K_LEN" depends on the tcpcrypt TEP in use, and is specified in Section 5.

To yield the _session ID_ required by TCP-ENO [I-D.ietf-tcpinc-tcpeno], tcpcrypt concatenates the first byte of the negotiated suboption (that is, including the "v" bit as transmitted by host B) with the bare session ID for a particular connection:

```
session ID = subopt-byte | SID
```

Given a session secret "ss", the two sides compute a series of master keys as follows:

```
mk[0] = CPRF (ss, CONST_REKEY, K_LEN)
mk[i] = CPRF (mk[i-1], CONST_REKEY, K_LEN)
```

Finally, each master key "mk" is used to generate keys for authenticated encryption for the "A" and "B" roles. Key "k_ab" is used by host A to encrypt and host B to decrypt, while "k_ba" is used by host B to encrypt and host A to decrypt.

```
k_ab = CPRF (mk, CONST_KEY_A, ae_keylen)
k_ba = CPRF (mk, CONST_KEY_B, ae_keylen)
```

The value "ae_keylen" depends on the authenticated-encryption algorithm selected, and is given under "Key Length" in Table 2.

After host B sends "Init2" or host A receives it, that host may immediately begin transmitting protected application data as described in Section 3.5.

3.4. Session caching

When two hosts have already negotiated session secret "ss[i-1]", they can establish a new connection without public-key operations using "ss[i]". Willingness to employ this facility is signalled by sending a SYN segment with a resumption suboption: an ENO suboption containing the negotiated TEP identifier from the original session and the flag "v = 1" (indicating variable-length data).

An active opener wishing to resume from a cached session may send a resumption suboption whose content is the nine-byte prefix of the associated bare session ID:

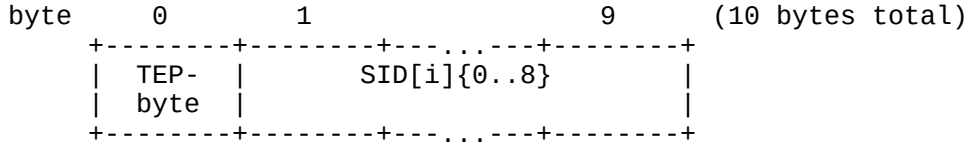


Figure 2: ENO suboption used to initiate session resumption. The TEP-byte contains a tcpcrypt TEP identifier and v = 1.

The active opener MUST use the lowest value of "i" that has not already been used to successfully negotiate resumption with the same host and for the same pre-session key "ss[0]".

In a particular SYN segment, a host SHOULD NOT send more than one resumption suboption, and MUST NOT send more than one resumption suboption with the same TEP identifier. But in addition to any resumption suboptions, an active opener MAY include non-resumption suboptions describing other key-agreement schemes it supports (in addition to that indicated by the TEP in the resumption suboption).

If the passive opener recognizes the prefix of "SID[i]" and knows "ss[i]", it SHOULD (with exceptions specified below) respond with an ENO option containing an _empty resumption suboption_ indicating the same key-exchange scheme; that is, a suboption whose initial byte gives the TEP identifier from host A's resumption suboption and sets "v = 1", but whose contents are empty. (The only way to encode this is as the last ENO suboption.)

Otherwise, the passive opener SHOULD attempt to negotiate fresh key exchange by responding with a single, non-resumption suboption with the same TEP as in the received resumption suboption, or with a TEP from another received suboption.

A host MUST ignore a resumption suboption if it has successfully negotiated resumption in the past, in either role, with the same "SID[i]". In the event that two hosts simultaneously send SYN segments to each other with the same "SID[i]", but the two segments are not part of a simultaneous open, both connections will have to revert to fresh key exchange. To avoid this limitation, implementations MAY choose to implement session caching such that a given pre-session key "ss[0]" is only used for either passive or active opens at the same host, not both.

In the case of simultaneous open where TCP-ENO is able to establish asymmetric roles, two hosts that simultaneously send SYN segments with resumption suboptions containing the same "SID[i]" may resume the associated session.

A host MUST NOT send, and upon receipt MUST ignore, an empty resumption suboption in a SYN-only segment.

After using "ss[i]" to compute "mk[0]", implementations SHOULD compute and cache "ss[i+1]" for possible use by a later session, then erase "ss[i]" from memory. Hosts SHOULD retain "ss[i+1]" until it is used or the memory needs to be reclaimed. Hosts SHOULD NOT write a cached "ss[i+1]" value to non-volatile storage.

When two hosts have previously negotiated a tcpcrypt session, either host may initiate session resumption regardless of which host was the active opener or played the "A" role in the previous session.

However, a given host must either encrypt with "k_ab" for all sessions derived from the same pre-session key "ss[0]", or with "k_ba". Thus, which keys a host uses to send segments is not affected by the role it plays in the current connection: it depends only on whether the host played the "A" or "B" role in the initial session.

Implementations that perform session caching MUST provide a means for applications to control session caching, including flushing cached session secrets associated with an ESTABLISHED connection or disabling the use of caching for a particular connection.

The session ID required by TCP-ENO and exposed to applications is constructed in the same way for resumed sessions as it is for fresh ones, as described above in Section 3.3. In particular, the first byte of the session ID is the first byte of the current connection's negotiated suboption, which means the byte will contain "v = 1"; and the remainder is "SID[i]", the bare session ID for the resumed session.

3.5. Data encryption and authentication

Following key exchange (or its omission via session caching), all further communication in a tcpcrypt-enabled connection is carried out within delimited _application frames_ that are encrypted and authenticated using the agreed keys.

This protection is provided via algorithms for Authenticated Encryption with Associated Data (AEAD). The particular algorithms that may be used are listed in Table 2. One algorithm is selected during the negotiation described in Section 3.3.

The format of an application frame is specified in Section 4.2. A sending host breaks its stream of application data into a series of chunks. Each chunk is placed in the "data" portion of a "plaintext" value, which is then encrypted to yield a frame's "ciphertext" field. Chunks must be small enough that the ciphertext (whose length depends on the AEAD cipher used, and is generally slightly longer than the plaintext) has length less than 2^16 bytes.

An "associated data" value (see Section 4.2.2) is constructed for the frame. It contains the frame's "control" field and the length of the ciphertext.

A "frame nonce" value (see Section 4.2.3) is also constructed for the frame (but not explicitly transmitted), containing an "offset" field whose integer value is the zero-indexed byte offset of the beginning of the current application frame in the underlying TCP datastream.

(That is, the offset in the framing stream, not the plaintext application stream.) Because it is strictly necessary for the security of the AEAD algorithm, an implementation MUST NOT ever transmit distinct frames with the same nonce value under the same encryption key. In particular, a retransmitted TCP segment MUST contain the same payload bytes for the same TCP sequence numbers, and a host MUST NOT transmit more than 2^64 bytes in the underlying TCP datastream (which would cause the "offset" field to wrap) before re-keying.

With reference to the "AEAD Interface" described in Section 2 of [RFC5116], tcpcrypt invokes the AEAD algorithm with the secret key "K" set to k_ab or k_ba, according to the host's role as described in Section 3.3. The plaintext value serves as "P", the associated data as "A", and the frame nonce as "N". The output of the encryption operation, "C", is transmitted in the frame's "ciphertext" field.

When a frame is received, tcpcrypt reconstructs the associated data and frame nonce values (the former contains only data sent in the clear, and the latter is implicit in the TCP stream), and provides these and the ciphertext value to the the AEAD decryption operation. The output of this operation is either "P", a plaintext value, or the special symbol FAIL. In the latter case, the implementation MUST either ignore the frame or abort the connection; but if it aborts, the implementation MUST raise an error condition distinct from the end-of-file condition.

3.6. TCP header protection

The "ciphertext" field of the application frame contains protected versions of certain TCP header values.

When "URGp" is set, the "urgent" value indicates an offset from the current frame's beginning offset; the sum of these offsets gives the index of the last byte of urgent data in the application datastream.

When "FINp" is set, it indicates that the sender will send no more application data after this frame. A receiver MUST ignore the TCP FIN flag and instead wait for "FINp" to signal to the local application that the stream is complete.

3.7. Re-keying

Re-keying allows hosts to wipe from memory keys that could decrypt previously transmitted segments. It also allows the use of AEAD ciphers that can securely encrypt only a bounded number of messages under a given key.

We refer to the two encryption keys (k_ab, k_ba) as a _key-set_. We refer to the key-set generated by mk[i] as the key-set with _generation number_ "i" within a session. Each host maintains a _current generation number_ that it uses to encrypt outgoing frames. Initially, the two hosts have current generation number 0.

When a host has just incremented its current generation number and has used the new key-set for the first time to encrypt an outgoing frame, it MUST set that frame's "rekey" field (see Section 4.2) to 1. It MUST set this field to zero in all other cases.

A host MAY increment its current generation number beyond the highest generation it knows the other side to be using. We call this action _initiating re-keying_.

A host SHOULD NOT initiate more than one concurrent re-key operation if it has no data to send; that is, it should not initiate re-keying with an empty application frame more than once while its record of the remote host's current generation number is less than its own.

On receipt, a host increments its record of the remote host's current generation number if and only if the "rekey" field is set to 1.

If a received frame's generation number is greater than the receiver's current generation number, the receiver MUST immediately increment its current generation number to match. After incrementing its generation number, if the receiver does not have any application data to send, it MUST send an empty application frame with the "rekey" field set to 1.

When retransmitting, implementations must always transmit the same bytes for the same TCP sequence numbers. Thus, a frame in a retransmitted segment MUST always be encrypted with the same key as when it was originally transmitted.

Implementations SHOULD delete older-generation keys from memory once they have received all frames they will need to decrypt with the old keys and have encrypted all outgoing frames under the old keys.

3.8. Keep-alive

Instead of using TCP Keep-Alives to verify that the remote endpoint is still responsive, tcpcrypt implementations SHOULD employ the re-keying mechanism, as follows. When necessary, a host SHOULD probe the liveness of its peer by initiating re-keying as described in Section 3.7, and then transmitting a new frame (with zero-length application data if necessary). A host receiving a frame whose key generation number is greater than its current generation number MUST

increment its current generation number and MUST immediately transmit a new frame (with zero-length application data, if necessary).

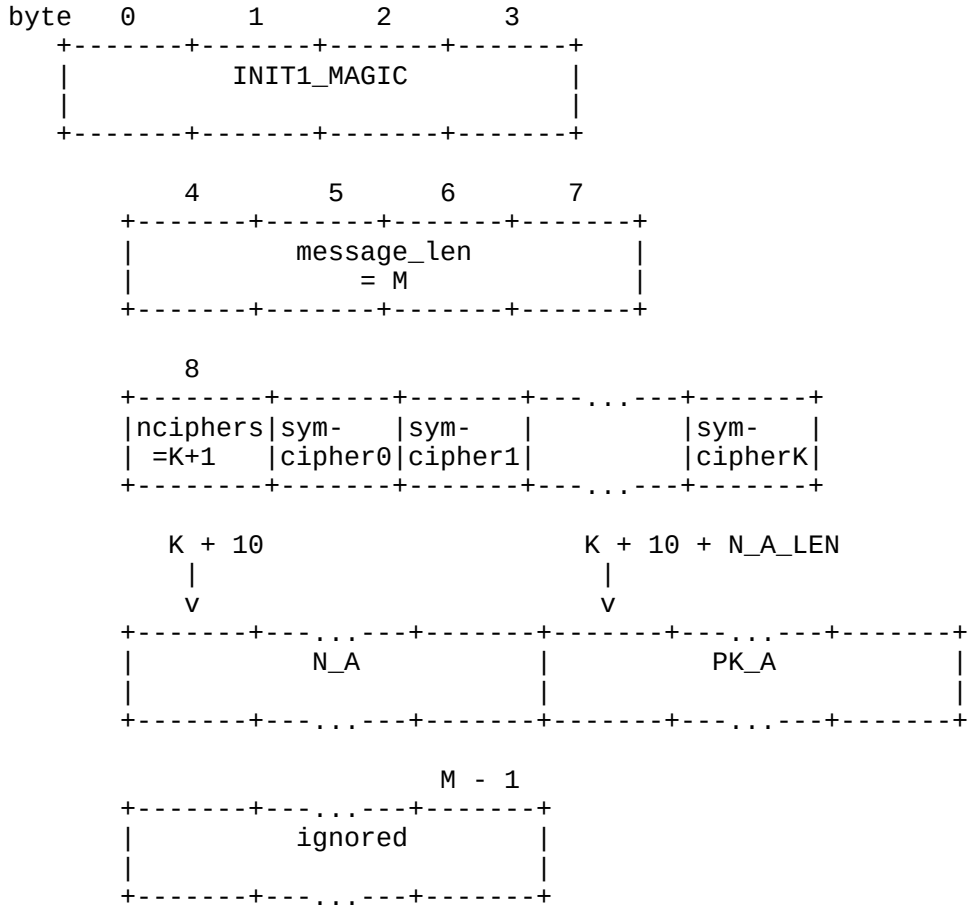
Implementations MAY use TCP Keep-Alives for purposes that do not require endpoint authentication, as discussed in Section 9.2.

4. Encodings

This section provides byte-level encodings for values transmitted or computed by the protocol.

4.1. Key exchange messages

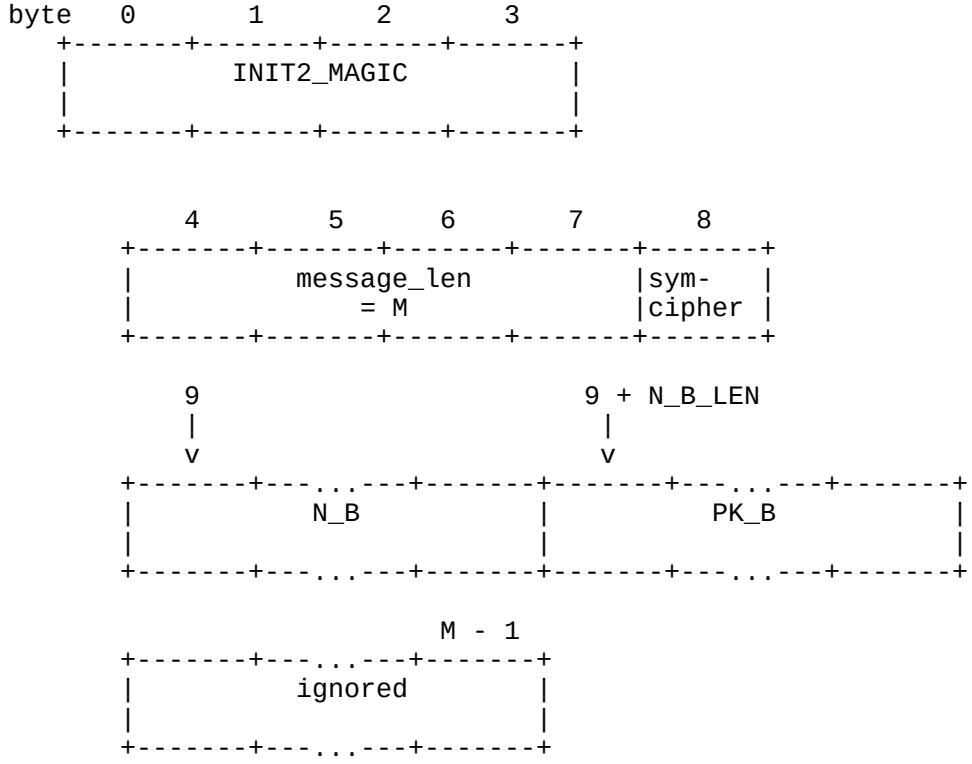
The "Init1" message has the following encoding:



The constant "INIT1_MAGIC" is defined in Table 3. The four-byte field "message_len" gives the length of the entire "Init1" message, encoded as a big-endian integer. The "nciphers" field contains an integer value that specifies the number of one-byte symmetric-cipher identifiers that follow. The "sym-cipher" bytes identify cryptographic algorithms in Table 2. The length "N_A_LEN" and the length of "PK_A" are both determined by the negotiated key-agreement scheme, as described in Section 5.

When sending "Init1", implementations of this protocol MUST omit the field "ignored"; that is, they must construct the message such that its end, as determined by "message_len", coincides with the end of the field "PK_A". When receiving "Init1", however, implementations MUST permit and ignore any bytes following "PK_A".

The "Init2" message has the following encoding:



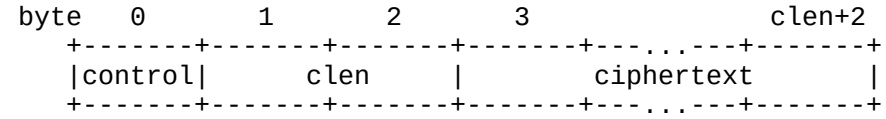
The constant "INIT2_MAGIC" is defined in Table 3. The four-byte field "message_len" gives the length of the entire "Init2" message, encoded as a big-endian integer. The "sym-cipher" value is a selection from the symmetric-cipher identifiers in the previously-

received "Init1" message. The length "N_B_LEN" and the length of "PK_B" are both determined by the negotiated key-agreement scheme, as described in Section 5.

When sending "Init2", implementations of this protocol MUST omit the field "ignored"; that is, they must construct the message such that its end, as determined by "message_len", coincides with the end of the "PK_B" field. When receiving "Init2", however, implementations MUST permit and ignore any bytes following "PK_B".

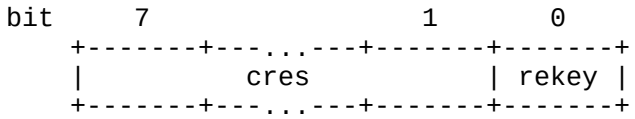
4.2. Application frames

An _application frame_ comprises a control byte and a length-prefixed ciphertext value:



The field "clen" is an integer in big-endian format and gives the length of the "ciphertext" field.

The byte "control" has this structure:

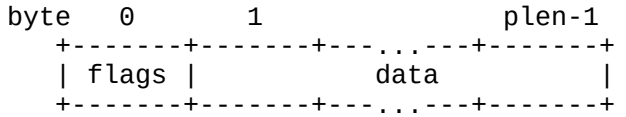


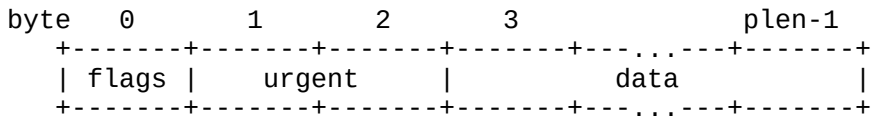
The seven-bit field "cres" is reserved; implementations MUST set these bits to zero when sending, and MUST ignore them when receiving.

The use of the "rekey" field is described in Section 3.7.

4.2.1. Plaintext

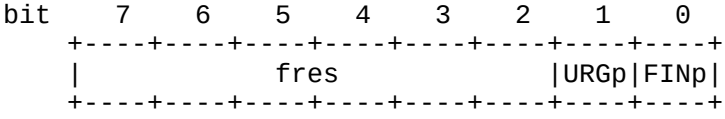
The "ciphertext" field is the result of applying the negotiated authenticated-encryption algorithm to a "plaintext" value, which has one of these two formats:





(Note that "clen" in the previous section will generally be greater than "plen", as the ciphertext produced by the authenticated-encryption scheme must both encrypt the application data and provide a way to verify its integrity.)

The "flags" byte has this structure:



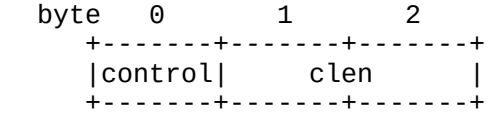
The six-bit value "fres" is reserved; implementations MUST set these six bits to zero when sending, and MUST ignore them when receiving.

When the "URGp" bit is set, it indicates that the "urgent" field is present, and thus that the plaintext value has the second structure variant above; otherwise the first variant is used.

The meaning of "urgent" and of the flag bits is described in Section 3.6.

4.2.2. Associated data

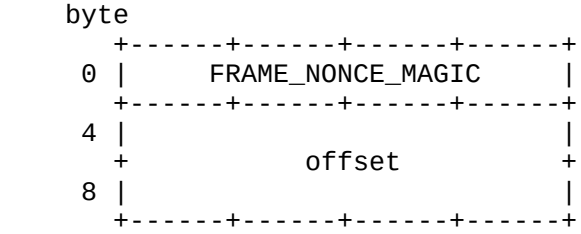
An application frame’s "associated data" (which is supplied to the AEAD algorithm when decrypting the ciphertext and verifying the frame’s integrity) has this format:



It contains the same values as the frame’s "control" and "clen" fields.

4.2.3. Frame nonce

Lastly, a "frame nonce" (provided as input to the AEAD algorithm) has this format:



The 4-byte magic constant is defined in Table 3. The 8-byte "offset" field contains an integer in big-endian format. Its value is specified in Section 3.5.

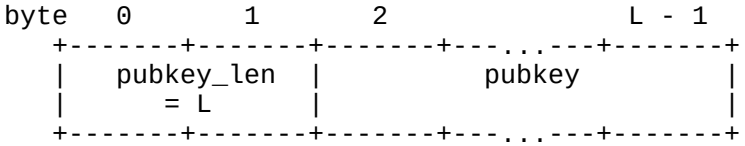
5. Key agreement schemes

The TEP negotiated via TCP-ENO may indicate the use of one of the key-agreement schemes named in Table 1. For example, "TCPCRYPT_ECDHE_P256" names the tcpcrypt protocol with key-agreement scheme ECDHE-P256.

All schemes listed there use HKDF-Expand-SHA256 as the CPRF, and these lengths for nonces and session keys:

```
N_A_LEN: 32 bytes
N_B_LEN: 32 bytes
K_LEN:   32 bytes
```

Key-agreement schemes ECDHE-P256 and ECDHE-P521 employ the ECSVDP-DH secret value derivation primitive defined in [ieee1363]. The named curves are defined in [nist-dss]. When the public-key values "PK_A" and "PK_B" are transmitted as described in Section 4.1, they are encoded with the "Elliptic Curve Point to Octet String Conversion Primitive" described in Section E.2.3 of [ieee1363], and are prefixed by a two-byte length in big-endian format:



Implementations SHOULD encode these "pubkey" values in "compressed format", and MUST accept values encoded in "compressed", "uncompressed" or "hybrid" formats.

Key-agreement schemes ECDHE-Curve25519 and ECDHE-Curve448 use the functions X25519 and X448, respectively, to perform the Diffie-Helman protocol as described in [RFC7748]. When using these ciphers, public-key values "PK_A" and "PK_B" are transmitted directly with no length prefix: 32 bytes for Curve25519, and 56 bytes for Curve448.

A tcpcrypt implementation MUST support at least the schemes ECDHE-P256 and ECDHE-P521, although system administrators need not enable them.

6. AEAD algorithms

Specifiers and key-lengths for AEAD algorithms are given in Table 2. The algorithms "AEAD_AES_128_GCM" and "AEAD_AES_256_GCM" are specified in [RFC5116]. The algorithm "AEAD_CHACHA20_POLY1305" is specified in [RFC7539].

7. IANA considerations

Tcpcrypt's TEP identifiers will need to be incorporated in IANA's TCP-ENO encryption protocol identifier registry, as follows:

cs	Spec name
0x21	TCPCRYPT_ECDHE_P256
0x22	TCPCRYPT_ECDHE_P521
0x23	TCPCRYPT_ECDHE_Curve25519
0x24	TCPCRYPT_ECDHE_Curve448

Table 1: TEP identifiers for use with tcpcrypt

A "tcpcrypt AEAD parameter" registry needs to be maintained by IANA as in the following table. The use of encryption is described in Section 3.5.

AEAD Algorithm	Key Length	sym-cipher
AEAD_AES_128_GCM	16 bytes	0x01
AEAD_AES_256_GCM	32 bytes	0x02
AEAD_CHACHA20_POLY1305	32 bytes	0x10

Table 2: Authenticated-encryption algorithms corresponding to sym-cipher specifiers in Init1 and Init2 messages.

8. Security considerations

Public-key generation, public-key encryption, and shared-secret generation all require randomness. Other tcpcrypt functions may also require randomness, depending on the algorithms and modes of operation selected. A weak pseudo-random generator at either host will compromise tcpcrypt's security. Many of tcpcrypt's cryptographic functions require random input, and thus any host implementing tcpcrypt MUST have access to a cryptographically-secure source of randomness or pseudo-randomness.

Most implementations will rely on system-wide pseudo-random generators seeded from hardware events and a seed carried over from the previous boot. Once a pseudo-random generator has been properly seeded, it can generate effectively arbitrary amounts of pseudo-random data. However, until a pseudo-random generator has been seeded with sufficient entropy, not only will tcpcrypt be insecure, it will reveal information that further weakens the security of the pseudo-random generator, potentially harming other applications. As required by TCP-ENO, implementations MUST NOT send ENO options unless they have access to an adequate source of randomness.

The cipher-suites specified in this document all use HMAC-SHA256 to implement the collision-resistant pseudo-random function denoted by "CPRF". A collision-resistant function is one on which, for sufficiently large L, an attacker cannot find two distinct inputs "K_1", "CONST_1" and "K_2", "CONST_2" such that "CPRF(K_1, CONST_1, L) = CPRF(K_2, CONST_2, L)". Collision resistance is important to assure the uniqueness of session IDs, which are generated using the CPRF.

All of the security considerations of TCP-ENO apply to tcpcrypt. In particular, tcpcrypt does not protect against active eavesdroppers unless applications authenticate the session ID. If it can be established that the session IDs computed at each end of the connection match, then tcpcrypt guarantees that no man-in-the-middle attacks occurred unless the attacker has broken the underlying

cryptographic primitives (e.g., ECDH). A proof of this property for an earlier version of the protocol has been published [tcpcrypt].

To gain middlebox compatibility, tcpcrypt does not protect TCP headers. Hence, the protocol is vulnerable to denial-of-service from off-path attackers. Possible attacks include desynchronizing the underlying TCP stream, injecting RST packets, and forging or suppressing rekey bits. These attacks will cause a tcpcrypt connection to hang or fail with an error. Implementations MUST give higher-level software a way to distinguish such errors from a clean end-of-stream (indicated by an authenticated "FINp" bit) so that applications can avoid semantic truncation attacks.

There is no "key confirmation" step in tcpcrypt. This is not required because tcpcrypt's threat model includes the possibility of a connection to an adversary. If key negotiation is compromised and yields two different keys, all subsequent frames will be ignored due to failed integrity checks, causing the application's connection to hang. This is not a new threat because in plain TCP, an active attacker could have modified sequence and acknowledgement numbers to hang the connection anyway.

Tcpcrypt uses short-lived public keys to provide forward secrecy. All currently specified key agreement schemes involve ECDHE-based key agreement, meaning a new key can be efficiently computed for each connection. If implementations reuse these parameters, they SHOULD limit the lifetime of the private parameters, ideally to no more than two minutes.

Attackers cannot force passive openers to move forward in their session caching chain without guessing the content of the resumption suboption, which will be difficult without key knowledge.

9. Design notes

9.1. Asymmetric roles

Tcpcrypt transforms a shared pseudo-random key (PRK) into cryptographic session keys for each direction. Doing so requires an asymmetry in the protocol, as the key derivation function must be perturbed differently to generate different keys in each direction. Tcpcrypt includes other asymmetries in the roles of the two hosts, such as the process of negotiating algorithms (e.g., proposing vs. selecting cipher suites).

9.2. Verified liveness

Many hosts implement TCP Keep-Alives [RFC1122] as an option for applications to ensure that the other end of a TCP connection still exists even when there is no data to be sent. A TCP Keep-Alive segment carries a sequence number one prior to the beginning of the send window, and may carry one byte of "garbage" data. Such a segment causes the remote side to send an acknowledgment.

Unfortunately, tcpcrypt cannot cryptographically verify Keep-Alive acknowledgments. Hence, an attacker could prolong the existence of a session at one host after the other end of the connection no longer exists. (Such an attack might prevent a process with sensitive data from exiting, giving an attacker more time to compromise a host and extract the sensitive data.)

Thus, tcpcrypt specifies a way to stimulate the remote host to send verifiably fresh and authentic data, described in Section 3.8.

The TCP keep-alive mechanism has also been used for its effects on intermediate nodes in the network, such as preventing flow state from expiring at NAT boxes or firewalls. As these purposes do not require the authentication of endpoints, implementations may safely accomplish them using either the existing TCP keep-alive mechanism or tcpcrypt’s verified keep-alive mechanism.

10. Acknowledgments

We are grateful for contributions, help, discussions, and feedback from the TCPINC working group, including Marcelo Bagnulo, David Black, Bob Briscoe, Jana Iyengar, Tero Kivinen, Mirja Kuhlewind, Yoav Nir, Christoph Paasch, Eric Rescorla, and Kyle Rose.

This work was funded by gifts from Intel (to Brad Karp) and from Google; by NSF award CNS-0716806 (A Clean-Slate Infrastructure for Information Flow Control); by DARPA CRASH under contract #N66001-10-2-4088; and by the Stanford Secure Internet of Things Project.

11. References

11.1. Normative References

[I-D.ietf-tcpinc-tcpeno]
Bittau, A., Boneh, D., Giffin, D., Handley, M., Mazieres, D., and E. Smith, "TCP-ENO: Encryption Negotiation Option", draft-ietf-tcpinc-tcpeno-06 (work in progress), October 2016.

[ieee1363]
"IEEE Standard Specifications for Public-Key Cryptography (IEEE Std 1363-2000)", 2000.

[nist-dss]
"Digital Signature Standard, FIPS 186-2", 2000.

[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>.

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

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <http://www.rfc-editor.org/info/rfc7539>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <http://www.rfc-editor.org/info/rfc7748>.

11.2. Informative References

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[tcpcrypt]
Bittau, A., Hamburg, M., Handley, M., Mazieres, D., and D. Boneh, "The case for ubiquitous transport-level encryption", USENIX Security , 2010.

Appendix A. Protocol constant values

Value	Name
0x01	CONST_NEXTK
0x02	CONST_SESSID
0x03	CONST_REKEY
0x04	CONST_KEY_A
0x05	CONST_KEY_B
0x15101a0e	INIT1_MAGIC
0x097105e0	INIT2_MAGIC
0x44415441	FRAME_NONCE_MAGIC

Table 3: Protocol constants

Authors' Addresses

Andrea Bittau
Google
345 Spear Street
San Francisco, CA 94105
US

Email: bittau@google.com

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

Mike Hamburg
Stanford University
353 Serra Mall, Room 475
Stanford, CA 94305
US

Email: mike@shiftleft.org

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

Quinn Slack
Stanford University
353 Serra Mall, Room 288
Stanford, CA 94305
US

Email: sqs@cs.stanford.edu

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

Email: eric.smith@kestrel.edu

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: May 4, 2017

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

October 31, 2016

TCP-ENO: Encryption Negotiation Option
draft-ietf-tcpinc-tcpeno-06

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.

Status of This Memo

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.

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, and

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 one among multiple possible TCP encryption protocols or _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.

2.1. 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 or _TEP_.

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. Provide a standard negotiation transcript through which TEPs 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. Terminology

We define the following terms, which are used throughout this document:

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 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 to negotiate one among multiple possible TCP encryption protocols or 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 indicating whether or not the application at each end knows it is using TCP-ENO.

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 [RFC0793]. There are two equivalent kinds of ENO option, shown in Figure 1. Section 10 specifies which of the two kinds is permissible and/or preferred.

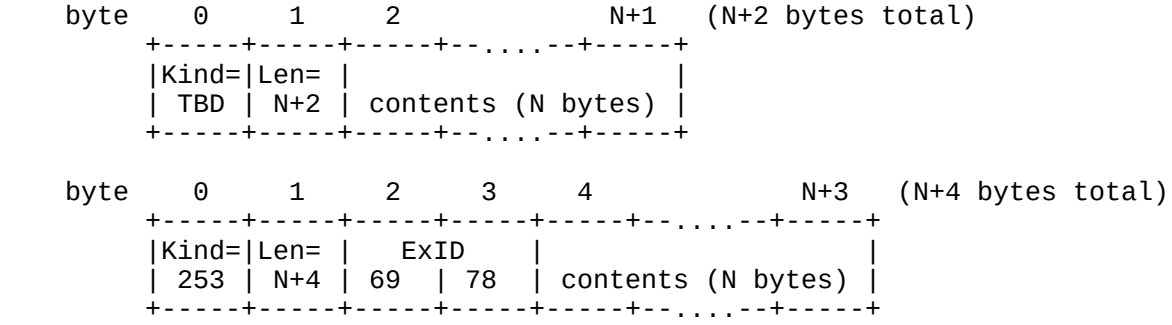


Figure 1: Two equivalent kinds of TCP-ENO option

The contents of an ENO option can take one of two forms. A SYN form, illustrated in Figure 2, appears only in SYN segments. A non-SYN form, illustrated in Figure 3, appears only in non-SYN segments. The SYN form of ENO acts as a container for zero or more suboptions, labeled "Opt_0", "Opt_1", ... in Figure 2. The non-SYN form, 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 ENO options. When a negotiated TEP does not assign such meaning, the contents of a non-SYN ENO option MUST be zero bytes.

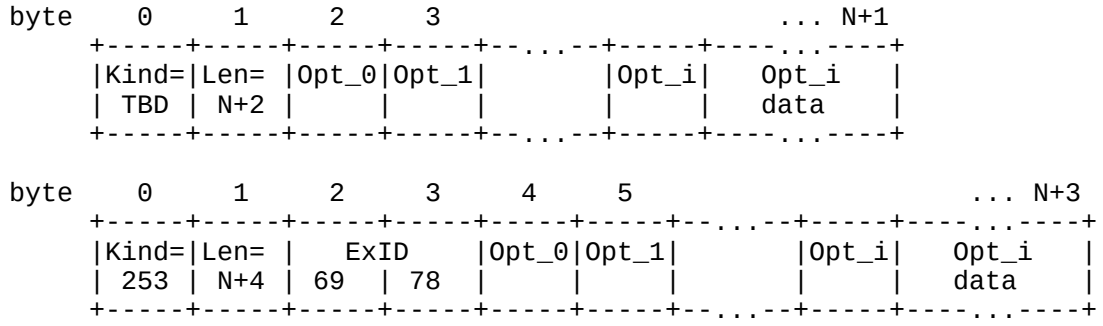


Figure 2: SYN form of ENO

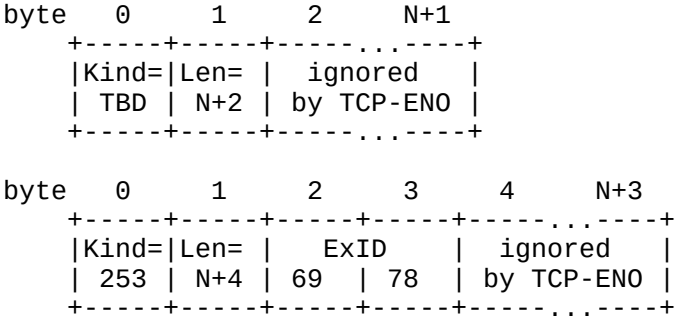
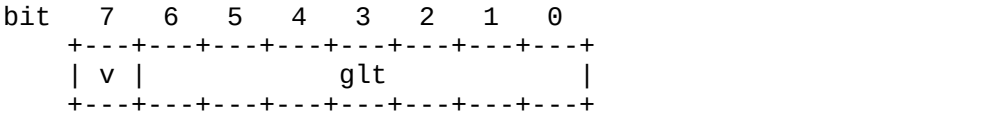


Figure 3: Non-SYN form of ENO, where N MAY be 0

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 7-bit value "glt" expresses one of:

- Global configuration data (discussed in Section 4.2),
- Suboption data length for the next suboption (discussed in Section 4.4), or
- 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", the initial suboption byte constitutes the entirety of the suboption and all information is expressed by the 7-bit "glt" value, which can be either a global suboption or TEP identifier. When "v = 1", it indicates a suboption with variable-length suboption data. Only TEP identifiers may have suboption data, not global suboptions. Hence, 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 identifer). 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 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 bitmask, illustrated in Figure 5.

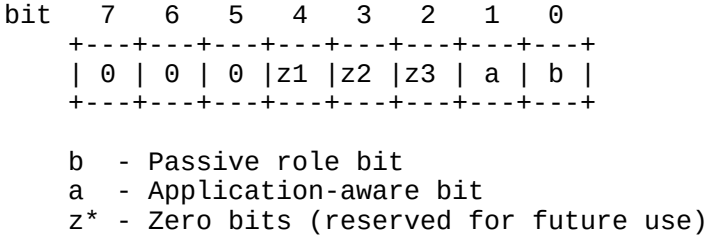


Figure 5: Format of the global suboption byte

The fields of the bitmask 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 set "b = 1" before initiating an active open. (Manual configuration of "b" is necessary for simultaneous open.)

a
: The application-aware bit "a" is an out-of-band signal indicating 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. Implementations MUST set this bit to 0 by default, and SHOULD 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 SHOULD furthermore support a _mandatory_ application-aware mode in which TCP-ENO is automatically disabled if the remote host does not set "a = 1".

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 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.

If both sides of a connection set "b = 1" (which can happen if the active opener misconfigures "b" before calling "connect"), or both sides set "b = 0" (which can happen with simultaneous open), then TCP-ENO MUST be disabled and the connection MUST 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). By default, suboption data 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. As an example, in Figure 2, the last suboption, "Opt_i", has suboption data and thus requires "v = 1"; however, the suboption data length can be 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 preceding 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).

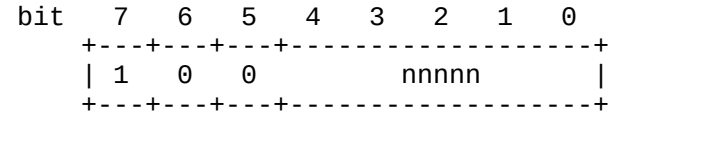


Figure 6: Format of a length byte

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.

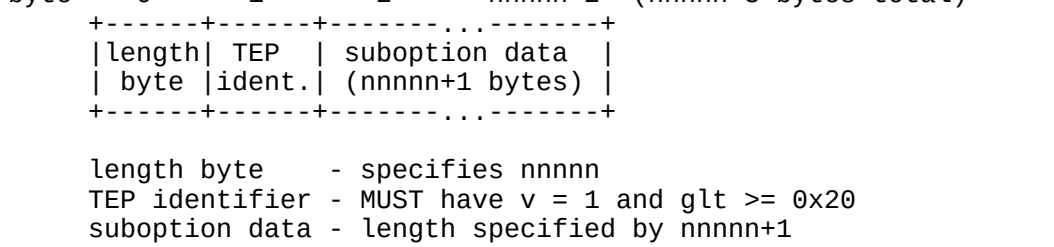


Figure 7: Suboption with length byte

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

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

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. The TCP header itself can only accommodate a maximum of 40 bytes of options per segment, however, so regardless of the length byte could not fit more than one suboption over 32 bytes. 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:

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, and

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

The _negotiated TEP_ is 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.

A passive opener (which is always host B) sees the remote host's SYN segment before constructing its own SYN-ACK. Hence, 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.

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.

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 a ENO option (or contains an ill-formed SYN-form ENO option),

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

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 be useful if middleboxes are dropping segments with the ENO option.) 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.

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 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 option 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 will govern a connection, hosts implementing ENO are REQUIRED to discard data from SYN+ENO segments when the SYN TEP does not govern the connection or when there is any ambiguity over the meaning of the SYN data. 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, or
- 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 governing the data is not the negotiated TEP (either because a different TEP was negotiated or because ENO failed to negotiate encryption), then the host MUST reset 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 reset the connection even if the SYN-ACK segment does not acknowledge the SYN data. The issue is that unacknowledged data may 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 reset, but applications MUST only use this mode when a higher-layer integrity check would anyway terminate a garbled connection.

To avoid unexpected connection resets, 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 resets a connection if ENO fails, and 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, TEPs need a way to reference a transcript of TCP-ENO's negotiation. In particular, a TEP MUST with high probability fail to reach key agreement between two honest endpoints if the TEP's selection resulted from tampering with the contents of SYN-form ENO options. (Of course, in the absence of endpoint authentication, two honest endpoints can still each end up talking to a man-in-the-middle attacker rather than to each other.)

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, TCP option length byte, and, for option kind 253, the bytes 69 and 78 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.

- 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 permit the negotiation of any encryption algorithms with significantly less than 128-bit security.

- 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 NOT depend on long-lived secrets for data confidentiality, as implementations SHOULD provide forward secrecy some bounded, short time after the close of a TCP connection. (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 high 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. A TEP MAY do so by cryptographically protecting 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. 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. 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 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.)

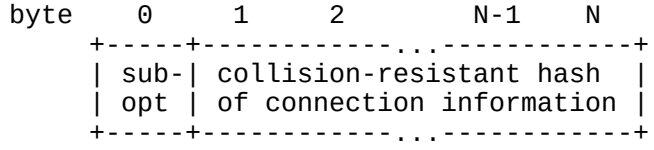


Figure 8: Format of a session ID

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, and
  - 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. 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 TEPs X and Y, and so 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 does not set the application-aware bit, or

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 network filtering

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. Hence, A does not receive an ENO option from B, disables ENO, and does not include a non-SYN-form ENO option when ACKing B'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<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 MUST 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. Design rationale

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

7.1. 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 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]. If SYN segments gain large options, it becomes possible to fit public keys or Diffie-Hellman parameters into SYN segments. Future TEPs can take advantage of this by performing key agreement directly within suboption data, both simplifying protocols and reducing the number of round trips required for connection setup.

If TCP gains large SYN option support, the 32-byte limit on length bytes may prove problematic. This draft intentionally aborts TCP-ENO if a length byte is followed by an octet in the range 0x00-0x9f. Any document updating TCP's option size limit can also enable larger

suboptions by updating this draft to assign 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 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.

7.2. 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 may drop ENO options in one direction only, a host must 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 must continue to include non-SYN form ENO options in segments until such point as 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.

7.3. 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.

7.4. 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 not 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 manually configure "b = 1". An alternate design might automatically break the symmetry to avoid this need for manual 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.)

7.5. Option kind sharing

This draft 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 TBD (or option kind 253 with ExID 0x454E) after the SYN exchange does not conflict with this document. Because in addition ENO guarantees at most one negotiated TEP per connection, TEPs will not conflict with one another or ENO if they use ENO's option kind for out-of-band signaling in non-SYN segments.

8. Experiments

This document has experimental status because TCP-ENO's viability depends on middlebox behavior that can only be determined _a posteriori_. Specifically, we must 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 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.

9. Security considerations

An obvious use case for TCP-ENO is opportunistic encryption--that is, encrypting some connections, but only where supported and without any kind of endpoint authentication. Opportunistic encryption 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.

Achieving stronger security with TCP-ENO requires verifying session IDs. Any application relying on ENO for communications security MUST incorporate session IDs into its endpoint authentication. By way of example, an authentication mechanism based on keyed digests (such Digest Access Authentication [RFC7616]) can be extended to include the role and session ID in the input of the keyed digest. To preserve backwards compatibility, applications MAY use the application-aware bit to negotiate the inclusion of session IDs in authentication.

Because TCP-ENO enables multiple different TEPs to coexist, security could potentially be only as strong as the weakest available TEP. In particular, if session IDs do not depend on the TCP-ENO transcript in

a strong way, an attacker can undetectably tamper with ENO options to force negotiation of a deprecated and vulnerable TEP. To avoid such problems, TEPs MUST compute session IDs using only well-studied and conservative hash functions. That way, even if other parts of a TEP are vulnerable, it is still 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 hence MUST be separately authenticated by the TEP.

10. IANA Considerations

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

Kind	Length	Meaning	Reference
TBD	N	Encryption Negotiation (TCP-ENO)	[RFC-TBD]

TCP Option Kind Numbers

Early implementations of TCP-ENO and a predecessor TCP encryption protocol made unauthorized use of TCP option kind 69.

[RFC-editor: please glue the following text to the previous paragraph iff TBD == 69, otherwise delete it.] 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.

[RFC-editor: please glue this to the previous paragraph regardless of the value of TBD.] More recent implementations used experimental option 253 per [RFC6994] with 16-bit ExID 0x454E, and SHOULD migrate to option TBD by default.

This document defines a 7-bit "glt" field in the range of 0x20-0x7f for which IANA shall maintain a new sub-registry entitled "TCP encryption protocol identifiers" under the "Transmission Control Protocol (TCP) Parameters" registry. The description of this registry should be interpreted with respect to the terminology defined in [RFC5226].

The intention is for IANA to grant registration requests for TEP identifiers in anticipation of a published RFC. Hence, a Specification is Required. However, to allow for implementation experience, identifiers should be allocated prior to the RFC being approved for publication. A Designated Expert appointed by the IESG area director shall approve allocations once it seems more likely than not that an RFC will eventually be published. The Designated Expert shall post a request to the TCPINC WG mailing list (or a successor designated by the Area Director) for comment and review, including an Internet-Draft. Before a period of 30 days has passed, the Designated Expert will either approve or deny the registration request and publish a notice of the decision to the TCPINC WG mailing list or its successor, as well as informing IANA. A denial notice must be justified by an explanation, and in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable should be provided.

The initial values of the TCP-ENO encryption protocol identifier registry are shown in Table 2.

Value	Meaning	Reference
0x20	Experimental Use	
0x21	TCPCRYPT_ECDHE_P256	[I-D.ietf-tcpinc-tcpcrypt]
0x22	TCPCRYPT_ECDHE_P521	[I-D.ietf-tcpinc-tcpcrypt]
0x23	TCPCRYPT_ECDHE_Curve25519	[I-D.ietf-tcpinc-tcpcrypt]
0x24	TCPCRYPT_ECDHE_Curve448	[I-D.ietf-tcpinc-tcpcrypt]
0x30	TCP-Use-TLS	[I-D.ietf-tcpinc-use-tls]

Table 2: TCP encryption protocol identifiers

11. Acknowledgments

We are grateful for contributions, help, discussions, and feedback from the TCPINC working group, including Marcelo Bagnulo, David Black, Bob Briscoe, Jana Iyengar, Tero Kivinen, Mirja Kuhlewind, Yoav Nir, Christoph Paasch, Eric Rescorla, Kyle Rose, and Joe Touch. This work was partially funded by DARPA CRASH and the Stanford Secure Internet of Things Project.

12. References

12.1. 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>.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <http://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, <http://www.rfc-editor.org/info/rfc7413>.

12.2. 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-tcpinc-tcpcrypt] Bittau, A., Boneh, D., Giffin, D., Hamburg, M., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-ietf-tcpinc-tcpcrypt-03 (work in progress), October 2016.

[I-D.ietf-tcpinc-use-tls] Rescorla, E., "Using TLS to Protect TCP Streams", draft-ietf-tcpinc-use-tls-01 (work in progress), May 2016.

[I-D.ietf-tcpm-tcp-edo]
Touch, J. and W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo-06 (work in progress), June 2016.

[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-05 (work in progress), October 2016.

[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>.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[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>.

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

Authors' Addresses

Andrea Bittau
Google
345 Spear Street
San Francisco, CA 94105
US

Email: bittau@google.com

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