Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 3, 2016

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

March 2, 2016

Interface Extensions for TCP-ENO
draft-bittau-tcpinc-api-01

Abstract

TCP-ENO negotiates encryption at the transport layer. It also defines a few parameters that are intended to be used or configured by applications. This document specifies operating system interfaces for access to these TCP-ENO parameters. We describe the interfaces in terms of socket options, the de facto standard API for adjusting per-connection behavior in TCP/IP, and sysctl, a popular mechanism for setting global defaults. Operating systems that lack socket or sysctl functionality can implement similar interfaces in their native frameworks, but should ideally adapt their interfaces from those presented in this document.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2016.

Table of Contents

1. Introduction

The TCP Encryption Negotiation Option (TCP-ENO) [I-D.ietf-tcpinc-tcpeno] permits hosts to negotiate encryption of a TCP connection. One of TCP-ENO's use cases is to encrypt traffic transparently, unbeknownst to legacy applications. Transparent encryption requires no changes to existing APIs. However, other use cases require applications to interact with TCP-ENO. In particular:

- Transparent encryption protects only against passive eavesdroppers. Stronger security requires applications to authenticate a _Session ID_ value associated with each encrypted connection.

- Applications that have been updated to authenticate Session IDs must somehow advertise this fact to peers in a backward-compatible way. TCP-ENO carries a two-bit "application-aware" status for

this purpose, but this status is not accessible through existing interfaces.

- Applications employing TCP's simultaneous open feature need a way to supply a symmetry-breaking "role-override" bit to TCP-ENO.

- System administrators and applications may wish to set and examine negotiation preferences, such as which encryption schemes (and perhaps versions) to enable and disable.

- Applications that perform their own encryption may wish to disable TCP-ENO entirely.

The remainder of this document describes an API through which systems can meet the above needs. The API extensions relate back to quantities defined by TCP-ENO.

2. API extensions

This section describes an API for per-connection options, followed by a discussion of system-wide configuration options.

2.1. Per-connection options

Application should access TCP-ENO options through the same mechanism they use to access other TCP configuration options, such as "TCP_NODELAY" [RFC0896]. With the popular sockets API, this mechanism consists of two socket options, "getsockopt" and "setsockopt", shown in Figure 1. Socket-based TCP-ENO implementations should define a set of new "option_name" values accessible at "level" "IPPROTO_TCP" (generally defined as 6, to match the IP protocol field).

```
int getsockopt(int socket, int level, int option_name,
               void *option_value, socklen_t *option_len);

int setsockopt(int socket, int level, int option_name,
               const void *option_value, socklen_t option_len);
```

Figure 1: Socket option API

Table 1 summarizes the new "option_name" arguments that TCP-ENO introduces to the socket option (or equivalent) system calls. For each option, the table lists whether it is read-only (R) or read-write (RW), as well as the type of the option's value. Read-write options, when read, always return the previously successfully written value or the default if they have not been written. Options of type "bytes" consist of a variable-length array of bytes, while options of

type "int" consist of a small integer with the exact range indicated in parentheses. We discuss each option in more detail below.

Option name	RW	Type
TCP_ENO_ENABLED	RW	int (-1 - 1)
TCP_ENO_SESSID	R	bytes
TCP_ENO_NEGSPEC	R	int (32 - 127)
TCP_ENO_SPECS	RW	bytes
TCP_ENO_SELF_AWARE	RW	int (0 - 3)
TCP_ENO_PEER_AWARE	R	int (0 - 3)
TCP_ENO_ROLEOVERRIDE	RW	int (0 - 1)
TCP_ENO_ROLE	R	int (0 - 1)
TCP_ENO_LOCAL_NAME	R	bytes
TCP_ENO_PEER_NAME	R	bytes
TCP_ENO_RAW	RW	bytes
TCP_ENO_TRANSCRIPT	R	bytes

Table 1: Suggested new IPPROTO_TCP socket options

The socket options must return errors under certain circumstances. These errors are mapped to three suggested error codes shown in Table 2. Most socket-based systems will already have constants for these errors. Non-socket systems should use existing error codes corresponding to the same conditions. "EINVAL" is the existing error returned when setting options on a closed socket. "EISCONN" corresponds to calling connect a second time, while "ENOTCONN" corresponds to requesting the peer address of an unconnected socket.

Symbol	Description
EINVAL	General error signifying bad parameters
EISCONN	Option no longer valid because socket is connected
ENOTCONN	Option not (yet) valid because socket not connected

Table 2: Suggested error codes

TCP_ENO_ENABLED When set to 0, completely disables TCP-ENO regardless of any other socket option settings except "TCP_ENO_RAW". When set to 1, enables TCP-ENO. If set to -1, use a system-wide default determined at the time of an "accept" or "connect" system call, as described in Section 2.2. This option must return an error ("EISCONN") after a SYN segment has already been sent.

TCP_ENO_SESSID Returns the session ID of the connection, as defined by the encryption spec in use. This option must return an error if encryption is disabled ("EINVAL"), the connection is not yet established ("ENOTCONN"), or the transport layer does not implement the negotiated spec ("EINVAL").

TCP_ENO_NEGSPEC Returns the 7-bit code point of the negotiated encryption spec for the current connection. As defined by TCP-ENO, the negotiated spec is the last valid suboption in the "B" host's SYN segment. This option must return an error if encryption is disabled ("EINVAL") or the connection is not yet established ("ENOTCONN").

TCP_ENO_SPECS Allows the application to specify an ordered list of encryption specs different from the system default list. If the list is empty, TCP-ENO is disabled for the connection. Each byte in the list specifies one suboption type from 0x20-0xff. The list contains no suboption data for variable-length suboptions, only the one-byte spec identifier. The high bit ("v") in these bytes is ignored unless future implementations of encryption specs assign it special meaning. The order of the list matters only for the host playing the "B" role. Implementations must return an error ("EISCONN") if an application attempts to set this option after the SYN segment has been sent. Implementations should return an error ("EINVAL") if any of the bytes are below 0x20 or are not implemented by the TCP stack.

TCP_ENO_SELF_AWARE The value is an integer from 0-3, allowing applications to specify the "aa" bits in the general suboption sent by the host. When listening on a socket, the value of this option applies to each accepted connection. The default value should be 0. Implementations must return an error ("EISCONN") if an application attempts to set this option after a SYN segment has been sent.

TCP_ENO_PEER_AWARE The value is an integer from 0-3 reporting the "aa" bits in the general suboption of the peer's segment. Implementations must return an error ("ENOTCONN") if an application attempts to read this value before the connection is established.

TCP_ENO_ROLEOVERRIDE The value is a bit (0 or 1), indicating the value of the "b" bit to set in the host's general suboption. The "b" bit breaks the symmetry of simultaneous open to assign a unique role "A" or "B" to each end of the connection. The host that sets the "b" bit assumes the "B" role (which in non-simultaneous open is by default assigned to the passive opener). Implementations must return an error ("EISCONN") for attempts to

set this option after the SYN segment has already been sent. The default value should be 0.

TCP_ENO_ROLE The value is a bit (0 or 1). TCP-ENO defines two roles, "A" and "B", for the two ends of a connection. After a normal three-way handshake, the active opener is "A" and the passive opener is "B". Simultaneous open uses the role-override bit to assign unique roles. This option returns 0 when the local host has the "A" role, and 1 when the local host has the "B" role. This call must return an error before the connection is established ("ENOTCONN") or if TCP-ENO has failed ("EINVAL").

TCP_ENO_LOCAL_NAME Returns the concatenation of the TCP_ENO_ROLE byte and the TCP_ENO_SESSID. This provides a unique name for the local end of the connection.

TCP_ENO_PEER_NAME Returns the concatenation of the negation of the TCP_ENO_ROLE byte and the TCP_ENO_SESSID. This is the same value as returned by TCP_ENO_LOCAL_NAME on the other host, and hence provides a unique name for the remote end of the connection.

TCP_ENO_RAW This option is for use by library-level implementations of encryption specs. It allows applications to make use of the TCP-ENO option, potentially including encryption specs not supported by the transport layer, and then entirely bypass any TCP-level encryption so as to encrypt above the transport layer. The default value of this option is a 0-byte vector, which disables RAW mode. If the option is set to any other value, it disables all other socket options described in this section except for TCP_ENO_TRANSCRIPT.

The value of the option is a raw ENO option contents (without the kind and length) to be included in the host’s SYN segment. In raw mode, the TCP layer considers negotiation successful when the two SYN segments both contain a suboption with the same encryption spec value "cs" >= 0x20. For an active opener in raw mode, the TCP layer automatically sends a two-byte minimal ENO option when negotiation is successful. Note that raw mode performs no sanity checking on the "v" bits or any suboption data, and hence provides slightly less flexibility than a true TCP-level implementation.

TCP_ENO_TRANSCRIPT Returns the negotiation transcript as specified by TCP-ENO. Implementations must return an error if negotiation failed ("EINVAL") or has not yet completed ("ENOTCONN").

2.2. System-wide options

In addition to these per-socket options, implementations should use "sysctl" or an equivalent mechanism to allow administrators to configure a default value for "TCP_ENO_SPECS", as well as default behavior for when "TCP_ENO_ENABLED" is -1. Table 3 provides a table of suggested parameters. The type "words" corresponds to a list of 16-bit unsigned words representing TCP port numbers (similar to the "baddynamic" sysctls that, on some operating systems, blacklist automatic assignment of particular ports). These parameters should be placed alongside most TCP parameters. For example, on BSD derived systems a suitable name would be "net.inet.tcp.eno_specs", while on Linux a more appropriate name would be "net.ipv4.tcp_eno_specs".

Name	Type
eno_specs	bytes
eno_enable_connect	int (0 - 1)
eno_enable_listen	int (0 - 1)
eno_bad_connect_ports	words
eno_bad_listen_ports	words

Table 3: Suggested sysctl values

"eno_specs" is simply a string of bytes, and provides the default value for the "TCP_ENO_SPECS" socket option. If "TCP_ENO_SPECS" is non-empty, the remaining sysctls determine whether to attempt TCP-ENO negotiation when the "TCP_ENO_ENABLED" option is -1 (the default), using the following rules.

- On active openers: If "eno_enable_connect" is 0, then TCP-ENO is disabled. If the remote port number is in "eno_bad_connect_ports", then TCP-ENO is disabled. Otherwise, the host attempts to use TCP-ENO.

- On passive openers: If "eno_enable_listen" is 0, then TCP-ENO is disabled. Otherwise, if the local port is in "eno_bad_listen_ports", then TCP-ENO is disabled. Otherwise, if the host receives an SYN segment with an ENO option containing compatible encryption specs, it attempts negotiation.

Because initial deployment may run into issues with middleboxes or incur slowdown for unnecessary double-encryption, sites may wish to blacklist particular ports. For example the following command:

```
sysctl net.inet.tcp.eno_bad_connect_ports=443,993
```

would disable ENO encryption on outgoing connections to ports 443 and 993 (which use application-layer encryption for HTTP and IMAP, respectively). If the per-socket "TCP_ENO_ENABLED" is not -1, it overrides the sysctl values.

On a server, running:

```
sysctl net.inet.tcp.eno_bad_listen_ports=443
```

makes it possible to disable TCP-ENO for incoming HTTPS connection without modifying the web server to set "TCP_ENO_ENABLED" to 0.

3. Examples

This section provides examples of how applications might authenticate session IDs. Authentication requires exchanging messages over the TCP connection, and hence is not backwards compatible with existing application protocols. To fall back to opportunistic encryption in the event that both applications have not been updated to authenticate the session ID, TCP-ENO provides the application-aware bits. To signal it has been upgraded to support application-level authentication, an application should set "TCP_ENO_SELF_AWARE" to 1 before opening a connection. An application should then check that "TCP_ENO_PEER_AWARE" is non-zero before attempting to send authenticators that would otherwise be misinterpreted as application data.

3.1. Cookie-based authentication

In cookie-based authentication, a client and server both share a cryptographically strong random or pseudo-random secret known as a "cookie". Such a cookie is preferably at least 128 bits long. To authenticate a session ID using a cookie, each host computes and sends the following value to the other side:

```
authenticator = PRF(cookie, local-name)
```

Here "PRF" is a pseudo-random function such as HMAC-SHA-256 [RFC6234]. "local-name" is the result of the "TCP_ENO_LOCAL_NAME" socket option. Each side must verify that the other side’s authenticator is correct. To do so, software obtains the remote host’s local name via the "TCP_ENO_PEER_NAME" socket option. Assuming the authenticators are correct, applications can rely on the TCP-layer encryption for resistance against active network attackers.

Note that if the same cookie is used in other contexts besides session ID authentication, appropriate domain separation must be

employed, such as prefixing "local-name" with a unique prefix to ensure "authenticator" cannot be used out of context.

3.2. Signature-based authentication

In signature-based authentication, one or both endpoints of a connection possess a private signature key the public half of which is known to or verifiable by the other endpoint. To authenticate itself, the host with a private key computes the following signature:

```
authenticator = Sign(PrivKey, local-name)
```

The other end verifies this value using the corresponding public key. Whichever side validates an authenticator in this way knows that the other side belongs to a host that possesses the appropriate signature key.

Once again, if the same signature key is used in other contexts besides session ID authentication, appropriate domain separation should be employed, such as prefixing "local-name" with a unique prefix to ensure "authenticator" cannot be used out of context.

4. Security considerations

The TCP-ENO specification discusses several important security considerations that this document incorporates by reference. The most important one, which bears reiterating, is that until and unless a session ID has been authenticated, TCP-ENO is vulnerable to an active network attacker, through either a downgrade or active man-in-the-middle attack.

Because of this vulnerability to active network attackers, it is critical that implementations return appropriate errors for socket options when TCP-ENO is not enabled. Equally critical is that applications must never use these socket options without checking for errors.

Applications with high security requirements that rely on TCP-ENO for security must either fail or fall back to application-layer encryption if TCP-ENO fails or session IDs authentication fails.

5. Acknowledgments

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

6. References

6.1. 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-01 (work in progress), February 2016.

6.2. Informative References

[RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <http://www.rfc-editor.org/info/rfc896>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 24, 2016

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

February 21, 2016

Cryptographic protection of TCP Streams (tcpcrypt)
draft-ietf-tcpinc-tcpcrypt-01

Abstract

This document specifies tcpcrypt, a cryptographic protocol that protects TCP payload data and is negotiated by means of 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 of option size restrictions, the protocol requires one additional one-way message latency to perform key exchange. However, this cost is avoided between two hosts that have recently established a previous tcpcrypt connection.

Status of This Memo

This Internet-Draft will expire on August 24, 2016.

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.
- Avoid unnecessary round trips.
- 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 TCP connections even when either end changes IP address.

3. Encryption protocol

This section describes the tcpcrypt protocol at an abstract level, so as to provide an overview and facilitate analysis. The next section specifies the byte formats of all messages.

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. 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.

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 with CONST is a single octet.

Once tcpcrypt has been successfully set up, we say the connection moves to an ENCRYPTING phase, where it employs an _authenticated

encryption mode_ to encrypt and integrity-protect all application data.

Note that 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. Thus, any host implementing tcpcrypt MUST have a cryptographically-secure source of randomness or pseudo-randomness.

3.2. Roles

Tcpcrypt transforms a single 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).

To establish roles for the hosts, tcpcrypt depends on 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, though an additional mechanism breaks the symmetry of simultaneous open. This document adopts the terms "A" and "B" to identify each end of a connection uniquely, following TCP-ENO's designation.

3.3. Protocol negotiation

Tcpcrypt also depends on TCP-ENO [I-D.ietf-tcpinc-tcpeno] to negotiate the use of tcpcrypt and a particular key agreement scheme. TCP-ENO negotiates an _encryption spec_ by means of suboptions embedded in SYN segments. Each suboption is identified by a byte consisting of a seven-bit _encryption spec identifier_ value, "cs", and a one-bit additional data indicator, "v". This document reserves and associates four "cs" values with tcpcrypt, as listed in Table 1; future standards can associate additional values with tcpcrypt.

A TCP connection MUST employ tcpcrypt and transition to the ENCRYPTING phase when and only when:

1. The TCP-ENO negotiated spec contains a "cs" value associated with tcpcrypt, and

2. The presence of variable-length data matches the suboption usage.

Specifically, when the "cs" value is "TCPCRYPT_RESUME", whose use is described in Section 3.5, there MUST be associated data (i.e., "v" MUST be 1). For all other "cs" values specified in this document, there MUST NOT be additional suboption data (i.e., "v" MUST be 0). Future "cs" values associated with tcpcrypt might or might not specify the use of associated data. Tcpcrypt implementations MUST ignore suboptions whose "cs" and "v" values do not agree as specified in this paragraph.

In normal usage, an active opener that wishes to negotiate the use of tcpcrypt will include an ENO option in its SYN segment; that option will include the tcpcrypt suboptions corresponding to the key-agreement schemes it is willing to enable, and possibly also a resumption suboption. 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 suboptions it supports, it MAY then attach an ENO option to its SYN-ACK segment, including _solely_ the suboption it wishes to enable.

Once two hosts have exchanged SYN segments, the _negotiated spec_ is the last spec 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 spec, hosts MUST disable TCP-ENO and tcpcrypt.

3.4. Key exchange

Following successful negotiation of a tcpcrypt spec, all further signaling is performed in the Data portion of TCP segments. If the negotiated spec is not TCPCRYPT_RESUME, the two hosts perform key exchange through two messages, INIT1 and INIT2, at the start of host A's and host B's data streams, respectively. INIT1 or INIT2 can 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 format of these messages is specified in detail in Section 4.1.

The parameters are defined as follows:

- 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 B from the sym-cipher-list sent by A.

- N_A, N_B: nonces chosen at random by A and B, respectively.

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

The pre-master secret (PMS) 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 PMS 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:

```
param  := { eno-transcript, init1, init2 }
PRK    := Extract (N_A, { param, PMS })
```

Above, "eno-transcript" is the protocol-negotiation transcript defined in TCP-ENO; "init1" and "init2" are the transmitted encodings of the INIT1 and INIT2 messages described in Section 4.1.

A series of "session secrets" and corresponding Session IDs are then computed 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 Session ID for the current connection, and will with overwhelming probability be unique for each individual TCP connection. The most computationally expensive part of the key exchange protocol is the public key cipher. 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 3.5. The CONST values are constants defined in Table 3. The K_LEN values depend on the tcpcrypt spec in use, and are specified in Section 6.

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

HKDF is not used directly for key derivation because tcpcrypt requires multiple expand steps with different keys. This is needed for forward secrecy, so that ss[n] can be forgotten once a session is established, and mk[n] can be forgotten once a session is rekeyed.

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 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.

3.5. 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]. A host wishing to request this facility will include in its SYN segment an ENO option whose last suboption contains the spec identifier TCPCRYPT_RESUME:

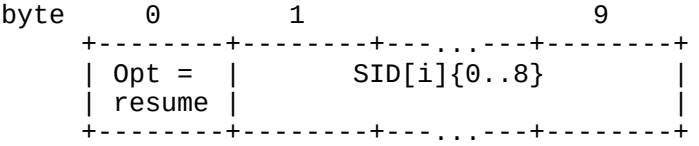


Figure 2: ENO suboption used to initiate session resumption

Above, the "resume" value is the byte whose lower 7 bits are TCPCRYPT_RESUME and whose top bit "v" is 1 (indicating variable-length data follows). The remainder of the suboption is filled with the first nine bytes of the Session ID SID[i].

A host SHOULD also include ENO suboptions describing the key-agreement schemes it supports in addition to a resume suboption, so as to fall back to full key exchange in the event that session resumption fails.

Which symmetric keys a host uses for transmitted segments is determined by its role in the original session ss[0]. It does not depend on the role it plays in the current session. For example, if a host had the "A" role in the first session, then it uses k_ab for sending segments and k_ba for receiving.

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 keep ss[i+1] around for a period of time 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.

It is an implementation-specific issue as to how long ss[i+1] should be retained if it is unused. If the passive opener evicts it from cache before the active opener does, the only cost is the additional ten bytes to send the resumption suboption in the next connection. The behavior then falls back to a normal public-key handshake.

The active opener MUST use the lowest value of "i" that has not already appeared in a resumption suboption exchanged with the same host and for the same pre-session seed.

If the passive opener recognizes SID[i] and knows ss[i], it SHOULD respond with an ENO option containing a dataless resumption suboption; that is, the suboption whose "cs" value is TCPCRYPT_RESUME and whose "v" bit is zero.

If the passive opener does not recognize SID[i], or SID[i] is not valid or has already been used, the passive opener SHOULD inspect any other ENO suboptions in hopes of negotiating a fresh key exchange as described in Section 3.4.

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 seed, or k_ba. Thus, which keys a host uses to send segments depends only whether the host played the

"A" or "B" role in the initial session that used ss[0]; it is not affected by which host was the active opener transmitting the SYN segment containing a resumption suboption.

A host MUST ignore a resumption suboption if it has previously sent or received one 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 public key cryptography. To avoid this limitation, implementations MAY choose to implement session caching such that a given pre-session key is only good 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.

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.

3.6. Data encryption and authentication

Following key exchange, 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 3.4.

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 frame’s "plaintext" value, which is then encrypted to yield the frame’s "ciphertext" field. Chunks must be small enough that the ciphertext (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 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.) As the security of the AEAD algorithm depends on this nonce being used to encrypt at most one distinct plaintext value, an implementation MUST NOT ever transmit distinct frames at the same location in the underlying TCP datastream.

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 3.4. 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 terminate the connection.

3.7. 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.

3.8. 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 the 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 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.

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.

3.9. Keep-alive

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.)

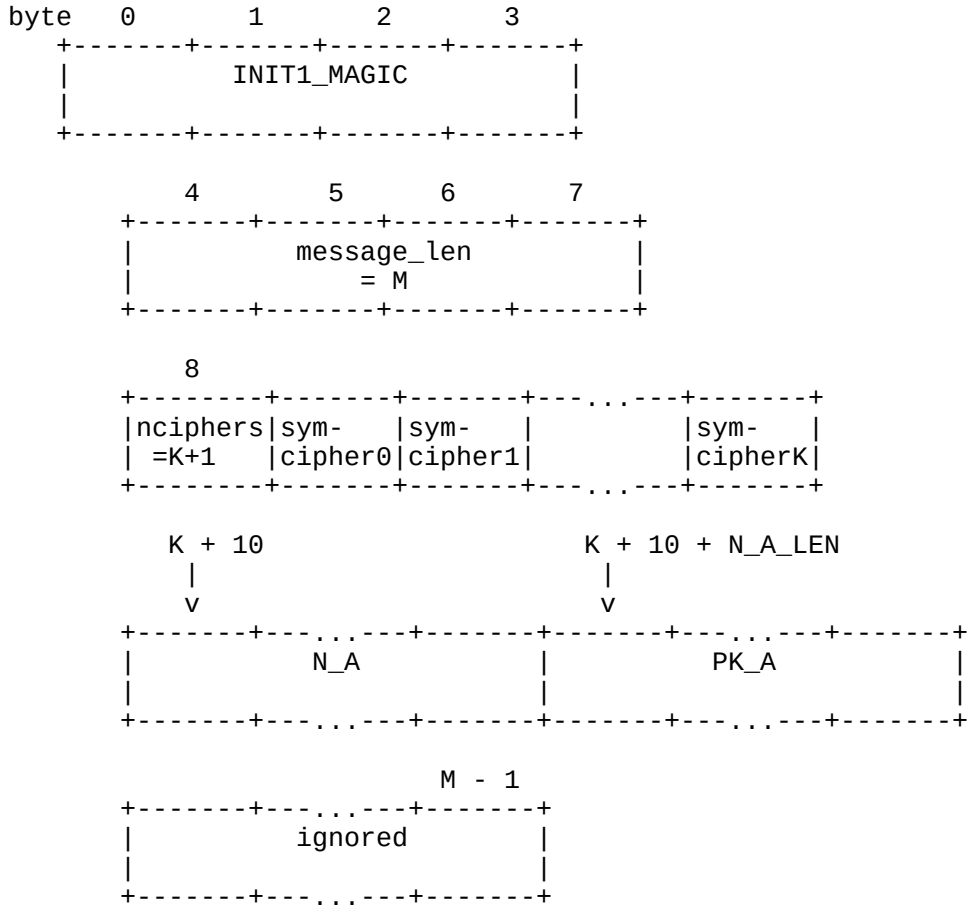
Instead of TCP Keep-Alives, tcpcrypt implementations SHOULD employ the re-keying mechanism to stimulate the remote host to send verifiably fresh and authentic data. When required, a host SHOULD probe the liveness of its peer by initiating re-keying as described in Section 3.8, and then transmitting a 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).

4. Encodings

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

4.1. Key exchange messages

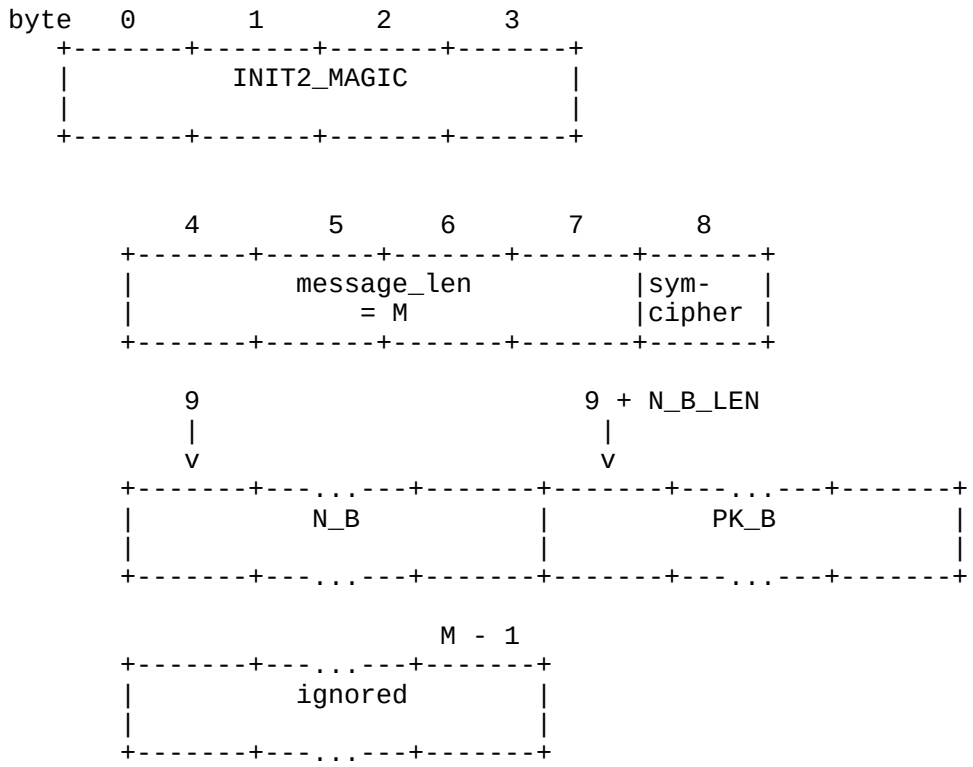
The INIT1 message has the following encoding:



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 6.

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 PK_A field. When receiving INIT1, however, implementations MUST permit and ignore any bytes following PK_A.

The INIT2 message has the following encoding:

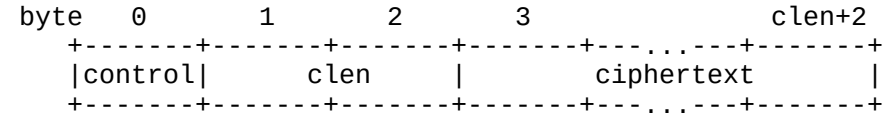


The constant INIT2_MAGIC is defined in Table 3. The four-byte field "message_len" gives the length of the entire INIT2 message, encoded as a big-endian integer. The "sym-cipher" value is a selection from the symmetric-cipher identifiers in the previously-received INIT1 message. The length N_B_LEN and the length of PK_B are both determined by the negotiated key-agreement scheme, as described in Section 6.

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.

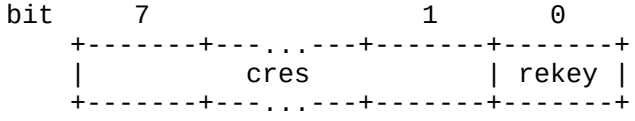
4.2. Application frames

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:

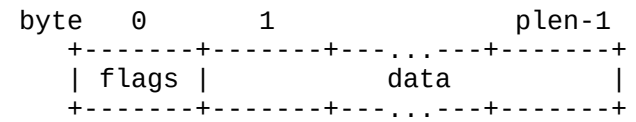


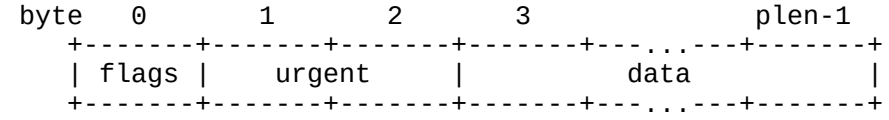
The seven-bit field "cres" is reserved; implementations MUST set these bits to zero when sending, and MUST ignore them when receiving.

The use of the "rekey" field is described in Section 3.8.

4.2.1. Plaintext

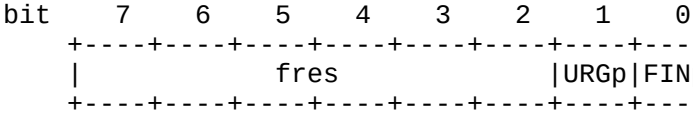
The "ciphertext" field is the result of applying the negotiated authenticated-encryption algorithm to a "plaintext" value, which has one of these two formats:





(Note that "clen" will generally be greater than "plen", as the authenticated-encryption scheme attaches an integrity "tag" to the encrypted input.)

The "flags" byte has this structure:



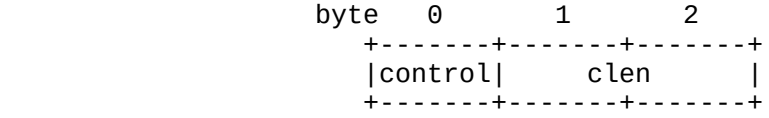
The six-bit value "fres" is reserved; implementations MUST set these six bits to zero when sending, and MUST ignore them when receiving.

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 3.7.

4.2.2. Associated data

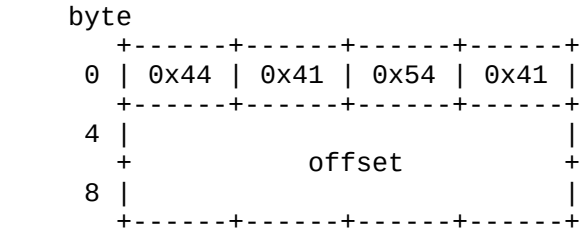
An application frame's "associated data" (which is supplied to the AEAD algorithm when decrypting the ciphertext and verifying the frame's integrity) has this format:



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 8-byte "offset" field contains an integer in big-endian format. Its value is specified in Section 3.6.

5. API extensions

Applications aware of tcpcrypt will need an API for interacting with the protocol. They can do so if implementations provide the recommended API for TCP-ENO. This section recommends several additions to that API, described in the style of socket options. However, these recommendations are non-normative:

The following options is read-only:

TCP_CRYPT_CONF: Returns the one-byte authenticated encryption algorithm in use by the connection (as specified in Table 2).

The following option is write-only:

TCP_CRYPT_CACHE_FLUSH: Setting this option to non-zero wipes cached session keys as specified in Section 3.5. Useful if application-level authentication discovers a man in the middle attack, to prevent the next connection from using session caching.

The following options should be readable and writable:

TCP_CRYPT_ACONF: Set of allowed symmetric ciphers and message authentication codes this host advertises in INIT1 messages.

TCP_CRYPT_BCONF: Order of preference of symmetric ciphers.

Finally, system administrators must be able to set the following system-wide parameters:

- Default TCP_CRYPT_ACONF value
- Default TCP_CRYPT_BCONF value
- Types, key lengths, and regeneration intervals of local host's short-lived public keys for implementations that do not use fresh ECDH parameters for each connection.

6. Key agreement schemes

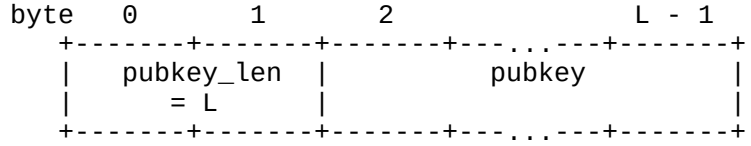
The encryption spec negotiated via TCP-ENO may indicate the use of one of the key-agreement schemes named in Table 1.

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.

7. 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].

8. Acknowledgments

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), and by DARPA CRASH under contract #N66001-10-2-4088.

9. IANA Considerations

Tcpcrypt’s spec identifiers ("cs" values) will need to be added to IANA’s ENO suboption registry, as follows:

cs	Spec name	Meaning
0x20	TCPCRYPT_RESUME	tcpcrypt session resumption
0x21	TCPCRYPT_ECDHE_P256	tcpcrypt with ECDHE-P256
0x22	TCPCRYPT_ECDHE_P521	tcpcrypt with ECDHE-P521
0x23	TCPCRYPT_ECDHE_Curve25519	tcpcrypt with ECDHE-Curve25519
0x24	TCPCRYPT_ECDHE_Curve448	tcpcrypt with ECDHE-Curve448

Table 1: cs values for use with tcpcrypt

A "tcpcrypt AEAD parameter" registry needs to be maintained by IANA as per the following table. The use of encryption is described in Section 3.6.

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.

10. Security considerations

It is worth reiterating just how crucial both the quality and quantity of randomness are to tcpcrypt’s security. 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. In the absence of secure hardware random generators, implementations MUST disable tcpcrypt after rebooting until the pseudo-random generator has been reseeded (usually by a bootup script) or sufficient entropy has been gathered.

Tcpcrypt guarantees that no man-in-the-middle attacks occurred if Session IDs match on both ends of a connection, unless the attacker has broken the underlying cryptographic primitives (e.g., ECDH). A proof has been published [tcpcrypt].

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.

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.

Similarly, tcpcrypt does not have a key confirmation step. Hence, an active attacker can cause a connection to hang, though this is possible even without tcpcrypt by altering sequence and ack numbers.

Tcpcrypt uses short-lived public key parameters to provide forward secrecy. All currently specified key agreement schemes involve ECDHE-based key agreement, meaning a new key can be chosen 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 hard without key knowledge.

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-01 (work in progress), February 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

Table 3: Protocol constants

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

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: August 22, 2016

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

February 19, 2016

TCP-ENO: Encryption Negotiation Option
draft-ietf-tcpinc-tcpeno-01

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 22, 2016.

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; however, such changes will ideally be transparent to applications that take advantage of TCP-level encryption. 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.

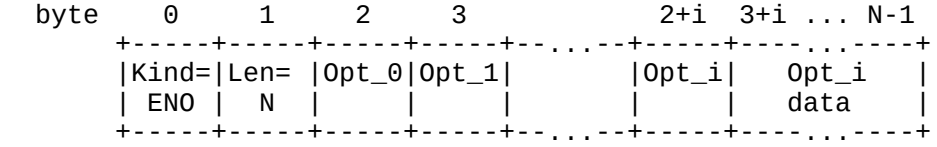


Figure 1: TCP-ENO option in SYN segment (MUST contain at least one suboption)

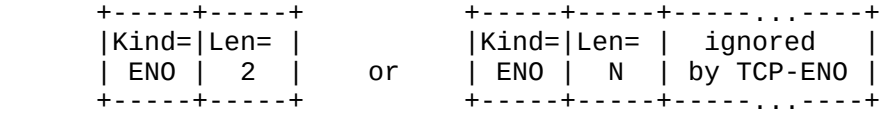
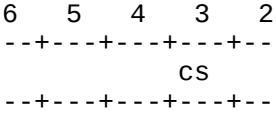


Figure 2: non-SYN TCP-ENO option in segment without SYN flag

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" specifies general parameters (discussed in Section 3.3), provides information about suboption length (discussed in Section 3.4), or indicates the willingness to use a specific encryption spec detailed in a separate document.



v - non-zero to indicate suboption data is present
cs - global configuration option or encryption spec identifier

Figure 3: Format of suboption byte

The high bit "v" in a suboption's first byte plays a role in determining whether a suboption carries additional data, and if so the length of that data. When "v = 0", a suboption carries no data

and consists simply of the seven-bit value "cs". When "v = 1" and "cs < 0x20", the suboption is a marker byte, specifying the length of the following suboption data, as discussed in Section 3.4. A marker byte MUST be followed by an encryption spec identifier with "v = 1" and one or more bytes of suboption data. In the absence of a marker byte, a suboption with "v = 1" extends to the end of the TCP option; in that case the length of the suboption data is determined by the total length of the TCP option. This design optimizes the common case that only the last suboption has any data, as no marker byte is needed under such circumstances. In Figure 1, "Opt_i" is the only option with data and there are no marker bytes; "Opt_i"'s total size is N-(2+i) bytes--one byte for "Opt_i" itself and N-(3+i) bytes for additional data.

Suboption data MAY be used for session caching, cipher suite negotiation, key exchange, or other purposes, as determined by the value of "cs".

A TCP SYN segment MUST contain at most one ENO TCP option. If a SYN segment contains multiple TCP options of kind ENO, the receiver MUST behave as though the segment contained no ENO options and disable encryption.

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, and values from 0x10-0x1f are reserved for possible future use by general suboptions. Values greater than or equal to 0x20 will be assigned to _spec identifiers_. When "v = 1", values in the range 0x00-0x1f become marker bytes while "cs" values greater than or equal to 0x20 MUST be followed by one or more bytes of suboption data. Implementations MUST ignore all unknown and reserved suboptions.

cs	Meaning
0x00-0x0f	General options (Section 3.3) and marker bytes (Section 3.4)
0x10-0x1f	Marker bytes and 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 with no special configuration, 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 first 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 role-override 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, "b", is the role-override bit. The least significant bit, "p", is 1 for a passive opener and 0 for an active opener. 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.

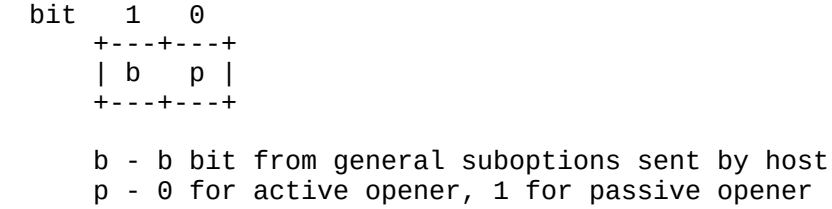


Figure 4: Role priority of an endpoint

Each host knows its own "p" bit is 0 if it sent a SYN segment without an ACK flag (a "SYN-only" segment), and is 1 otherwise. Each host estimates the other host's "p" bit as 0 if it receives a SYN-only segment, and as 1 otherwise. An important subtlety is that because of a lost or delayed SYN-only segment, one of the two hosts in a simultaneous open may incorrectly assume the other host has "p" set to 1. In the event that the two hosts set different "b" bits, no harm is done as the "b" bit overrides the "p" bit for role selection. In the event that both "b" bits are the same, both hosts have the

same role priority and TCP-ENO MUST be aborted. Fortunately, at least one host will always detect a priority tie before sending a SYN-ACK segment, and hence will force TCP-ENO to abort by sending its SYN-ACK without an ENO option.

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 in a three-way handshake 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 need to send a non-SYN ENO option.

- Under simultaneous open, each host sends both a SYN-only segment and a SYN-ACK segment. In this case, if negotiation succeeds, ENO options must be identical in each hosts's SYN-only and SYN-ACK segment. If negotiation fails (for instance because of a tie in role priority), then a host detecting this failure MUST send a SYN-ACK segment without an 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 contents or lack of suboption data in the two SYN segments is well-defined by the corresponding encryption spec. Specs MAY require, allow, or disallow suboption data in each of 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 because it is most space efficient to place a variable-length suboption at the end of a TCP-ENO option. When using this optimization, favoring the last suboption favors the spec with suboption data.

When possible, host B SHOULD send only one spec identifier (suboption with "cs" in the range 0x20-0x7f), 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, 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 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.

Note that with a regular three-way handshake (meaning no simultaneous open), the mandatory ENO option in an active opener's first ACK segment MAY contain spec-specific data, as shown on the right in Figure 2. Such data is not part of the TCP-ENO negotiation transcript. Hence, an encryption spec MUST take steps to authenticate any data it embeds in non-SYN ENO options.

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 role-override 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. A receiver SHOULD disable TCP-ENO upon receipt of a SYN segment with multiple general suboptions.

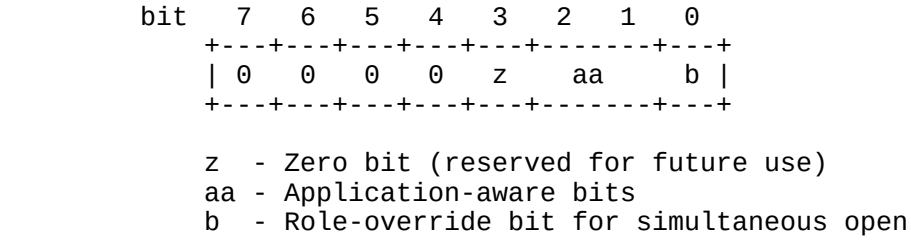


Figure 9: Format of the general option byte

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

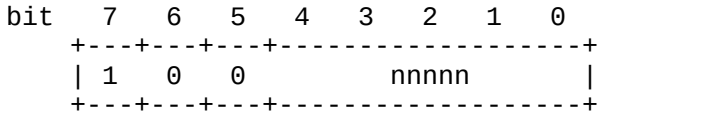
Table 2: Meaning of the two application-aware bits

b This is the role-override 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. Specifying suboption data length

When a TCP-ENO option contains multiple suboptions with data, or when a suboption other than the last one has data, it is necessary to specify the length of the suboption so that the receiver knows at what point to start parsing the next suboption. The length of suboption data can be specified by placing a marker byte immediately before a suboption.

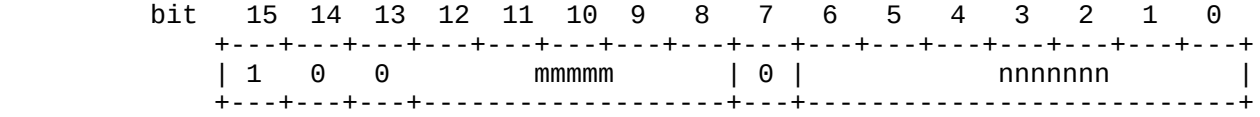


nnnnn - 5-bit value encodes (length - 1)

Figure 10: Format of a marker byte

Figure 10 shows the format of a marker byte. It encodes a 5-bit value "nnnnn". Adding one to this value specifies the length of the suboption data. Hence a marker byte can designate a suboption carrying anywhere from 1 to 32 bytes of data (inclusive). Note that the length does not count the marker byte or suboption byte, only suboption data following the suboption byte. For instance, marker byte 0x9f would be followed by a suboption byte and 32 bytes of suboption data, together occupying a total of 34 bytes within an ENO TCP option.

The suboption following a marker byte MUST always have "v = 1", and must always contain at least one byte of suboption data.



mmmmm - 5 most significant bits of 12-bit value (length - 1)
nnnnnnn - 7 least significant bits of 12-bit value (length - 1)

Figure 11: Format of a marker word

If the octet following a marker byte has the high bit clear (meaning "v = 0"), then the marker byte and following octet together are interpreted as a marker word, as shown in Figure 11. The length thus encoded does not count the marker word or suboption byte, only the suboption data following the suboption byte. Marker words are primarily intended for use in conjunction with future TCP extensions for large options. (Such an extention would need to overcome both TCP's 40-byte option limit and the single-byte TCP option length to make use of all 12 bits of length.)

If a marker byte or word in a received SYN segment indicates that a TCP-ENO option would extend beyond the end of the TCP option, the receiver MUST behave as though the received SYN segment contains no TCP-ENO options and fall back to unencrypted TCP.

3.5. 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. 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, the transcript unambiguously delimits A's and B's ENO options.

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, with two exceptions for an active opener. First, 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. Second, an active opener performing simultaneous open MAY include no TCP-ENO option in its SYN-ACK if the two hosts' SYN-only segments contain incompatible TCP-ENO options (for instance because role negotiation failed).

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 12 shows the resulting format. This format is designed for spec authors 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.

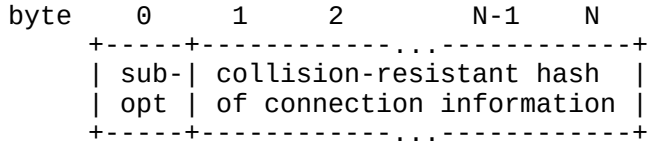


Figure 12: Format of a session ID

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 3.5 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. 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. Should multiple session IDs be required, it might be necessary to require all encryption specs to provide a feature similar to TLS exporters [RFC5705].

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-04 (work in progress), November 2015.

[I-D.ietf-tls-tls13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-11 (work in progress), December 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-03 (work in progress), October 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, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <http://www.rfc-editor.org/info/rfc5705>.

[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>.

[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