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.

Copyright Notice

Table of Contents

1. Requirements language 3
2. Introduction 3
3. Encryption protocol 3
3.1. Cryptographic algorithms 4
3.2. Roles 5
3.3. Protocol negotiation 5
3.4. Key exchange 6
3.5. Session caching 8
3.6. Data encryption and authentication 10
3.7. TCP header protection 11
3.8. Re-keying 11
3.9. Keep-alive 12
4. Encodings 13
4.1. Key exchange messages 13
4.2. Application frames 15
4.2.1. Plaintext 16
4.2.2. Associated data 17

4.2.3. Frame nonce 17
5. API extensions 17
6. Key agreement schemes 18
7. AEAD algorithms 19
8. Acknowledgments 19
9. IANA Considerations 19
10. Security considerations 20
11. References 21
11.1. Normative References 21
11.2. Informative References 22
Appendix A. Protocol constant values 22
Authors’ Addresses 23

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:

```
byte     0        1                  9
     +--------+--------+---...---+--------+
     | Opt =  |       SID[i]{0..8}        |
     | 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:

```
       byte   0       1       2       3
          +-------+-------+-------+-------+
          |          INIT1_MAGIC          |
          |                               |
          +-------+-------+-------+-------+

                  4        5      6       7
              +-------+-------+-------+-------+
              |          message_len          |
              |              = M              |
              +-------+-------+-------+-------+

                  8
              +--------+-------+-------+---...---+-------+
              |nciphers|sym-   |sym-   |         |sym-   |
              | =K+1   |cipher0|cipher1|         |cipherK|
              +--------+-------+-------+---...---+-------+

                 K + 10                    K + 10 + N_A_LEN
                  |                          |
                  v                          v
              +-------+---...---+-------+-------+---...---+-------+
              |           N_A           |          PK_A           |
              |                         |                         |
              +-------+---...---+-------+-------+---...---+-------+

                                  M - 1
              +-------+---...---+-------+
              |          ignored        |
              |                         |
              +-------+---...---+-------+
```

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:

```
       byte   0       1       2       3
          +-------+-------+-------+-------+
          |          INIT2_MAGIC          |
          |                               |
          +-------+-------+-------+-------+


                  4        5      6       7       8
              +-------+-------+-------+-------+-------+
              |          message_len          |sym-   |
              |              = M              |cipher |
              +-------+-------+-------+-------+-------+

                  9                        9 + N_B_LEN
                  |                         |
                  v                         v
              +-------+---...---+-------+-------+---...---+-------+
              |           N_B           |          PK_B           |
              |                         |                         |
              +-------+---...---+-------+-------+---...---+-------+

                                  M - 1
              +-------+---...---+-------+
              |          ignored        |
              |                         |
              +-------+---...---+-------+
```

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:

```
          byte   0       1       2       3               clen+2
             +-------+-------+-------+-------+---...---+-------+
             |control|      clen     |        ciphertext       |
             +-------+-------+-------+-------+---...---+-------+
```

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

The byte "control" has this structure:

```
                  bit     7                 1       0
                      +-------+---...---+-------+-------+
                      |          cres           | rekey |
                      +-------+---...---+-------+-------+
```

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:

```
          byte   0       1               plen-1
             +-------+-------+---...---+-------+
             | flags |           data          |
             +-------+-------+---...---+-------+


          byte   0       1       2       3              plen-1
             +-------+-------+-------+-------+---...---+-------+
             | flags |    urgent     |          data           |
             +-------+-------+-------+-------+---...---+-------+
```

(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:

```
               bit    7    6    5    4    3    2    1    0
                   +----+----+----+----+----+----+----+----+
                   |            fres             |URGp|FINp|
                   +----+----+----+----+----+----+----+----+
```

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:

```
byte   0       1       2
   +-------+-------+-------+
   |control|     clen      |
   +-------+-------+-------+
```

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:

```
byte
   +------+------+------+------+
 0 | 0x44 | 0x41 | 0x54 | 0x41 |
   +------+------+------+------+
 4 |                           |
   +          offset           +
 8 |                           |
   +------+------+------+------+
```

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:

```
              byte   0       1       2               L - 1
                 +-------+-------+-------+---...---+-------+
                 |   pubkey_len  |          pubkey         |
                 |      = L      |                         |
                 +-------+-------+-------+---...---+-------+
```

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