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

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

October 31, 2016

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

Abstract

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

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

Copyright Notice

Table of Contents

1. Requirements language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

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

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

3. Encryption protocol

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

3.1. Cryptographic algorithms

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

- A _key agreement scheme_ is used with a short-lived public key to agree upon a shared secret.

- An _extract function_ is used to generate a pseudo-random key from some initial keying material, typically the output of the key agreement scheme. The notation Extract(S, IKM) denotes the output of the extract function with salt S and initial keying material IKM.

- A _collision-resistant pseudo-random function (CPRF)_ is used to generate multiple cryptographic keys from a pseudo-random key, typically the output of the extract function. We use the notation CPRF(K, CONST, L) to designate the output of L bytes of the pseudo-random function identified by key K on CONST.

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

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

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

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

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

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

3.2. Protocol negotiation

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

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

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

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

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

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

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

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

3.3. Key exchange

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

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

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

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

The parameters are defined as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.4. Session caching

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

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

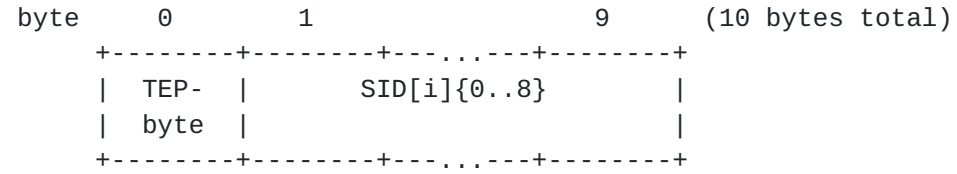


Figure 2: ENO suboption used to initiate session resumption. The TEP-byte contains a tcpcrypt TEP identifier and v = 1.

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

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

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

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

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

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

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

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

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

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

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

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

3.5. Data encryption and authentication

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

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

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

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

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

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

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

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

3.6. TCP header protection

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

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

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

3.7. Re-keying

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

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

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

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

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

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

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

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

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

3.8. Keep-alive

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

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

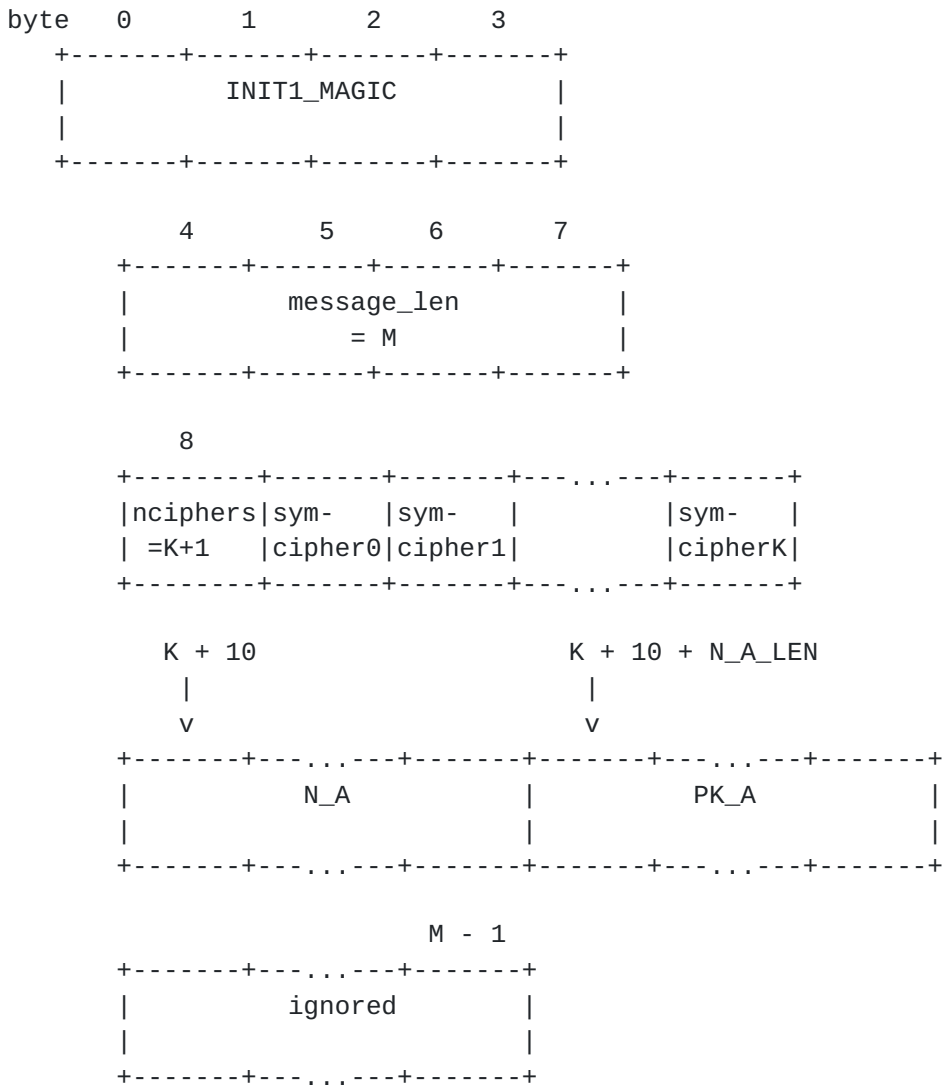
Implementations MAY use TCP Keep-Alives for purposes that do not require endpoint authentication, as discussed in Section 9.2.

4. Encodings

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

4.1. Key exchange messages

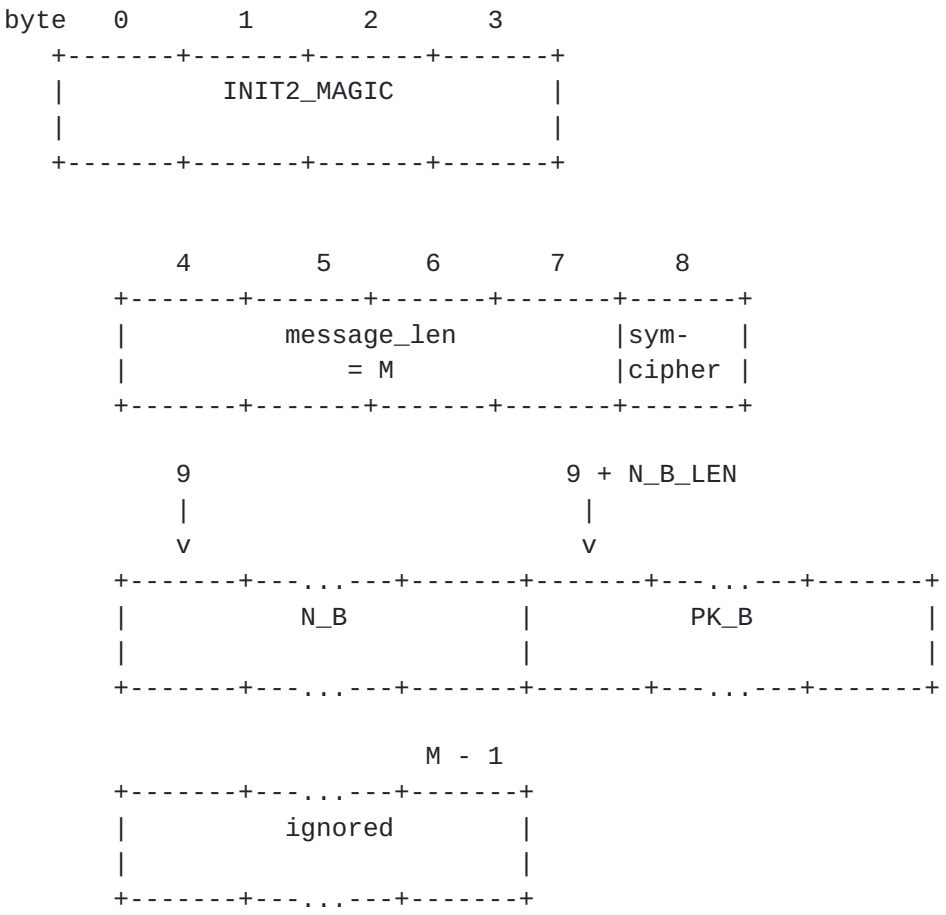
The "Init1" message has the following encoding:



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

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

The "Init2" message has the following encoding:



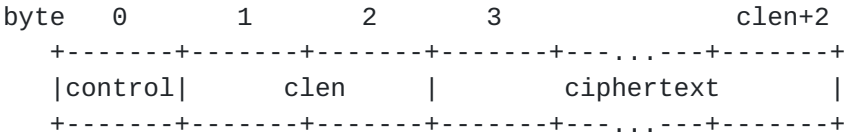
The constant "INIT2_MAGIC" is defined in Table 3. The four-byte field "message_len" gives the length of the entire "Init2" message, encoded as a big-endian integer. The "sym-cipher" value is a selection from the symmetric-cipher identifiers in the previously-

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

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

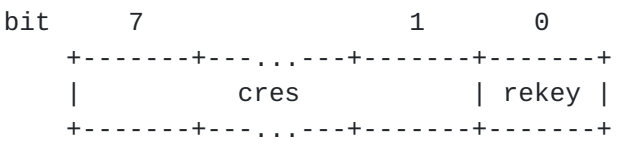
4.2. Application frames

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



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

The byte "control" has this structure:

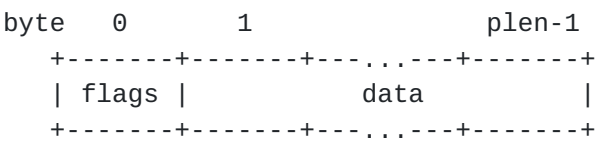


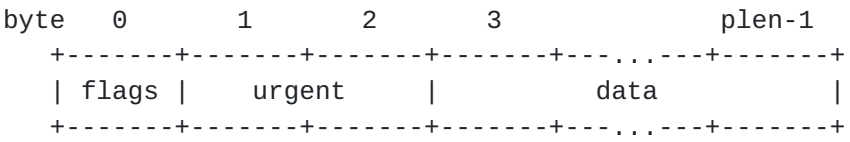
The seven-bit field "cres" is reserved; implementations MUST set these bits to zero when sending, and MUST ignore them when receiving.

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

4.2.1. Plaintext

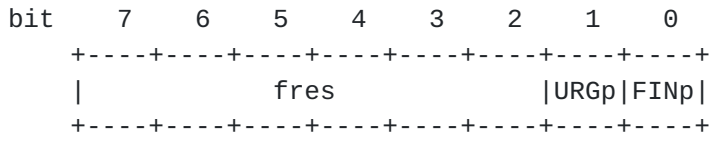
The "ciphertext" field is the result of applying the negotiated authenticated-encryption algorithm to a "plaintext" value, which has one of these two formats:





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

The "flags" byte has this structure:



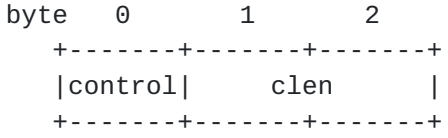
The six-bit value "fres" is reserved; implementations MUST set these six bits to zero when sending, and MUST ignore them when receiving.

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

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

4.2.2. Associated data

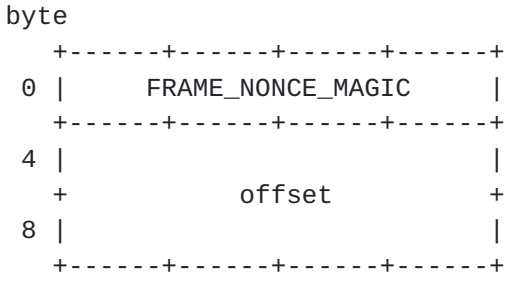
An application frame's "associated data" (which is supplied to the AEAD algorithm when decrypting the ciphertext and verifying the frame's integrity) has this format:



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

4.2.3. Frame nonce

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



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

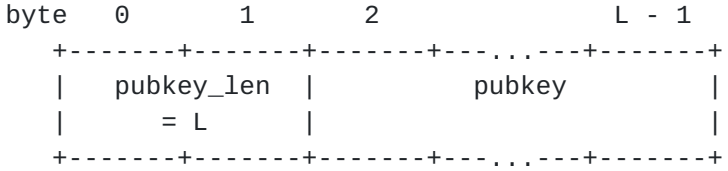
5. Key agreement schemes

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

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

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

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



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

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

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

6. AEAD algorithms

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

7. IANA considerations

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

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

Table 1: TEP identifiers for use with tcpcrypt

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

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

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

8. Security considerations

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

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

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

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

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

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

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

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

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

9. Design notes

9.1. Asymmetric roles

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

9.2. Verified liveness

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

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

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

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

10. Acknowledgments

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

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

11. References

11.1. Normative References

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

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

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

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

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

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

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

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

11.2. Informative References

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

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

Appendix A. Protocol constant values

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

Table 3: Protocol constants

Authors' Addresses

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

Email: bittau@google.com

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

Email: dabo@cs.stanford.edu

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

Email: dbg@scs.stanford.edu

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

Email: mike@shiftleft.org

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

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

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

Email: dm@uun.org

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

Email: sqs@cs.stanford.edu

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

Email: eric.smith@kestrel.edu