Network Working Group   C. Wood, Ed.
Internet-Draft   Apple Inc.
Intended status: Informational   T. Enghardt
Expires: March 31, 2020   TU Berlin
                                                                 T. Pauly
                                                               Apple Inc.
                                                               C. Perkins
                                                     University of Glasgow
                                                                  K. Rose
                                                 Akamai Technologies, Inc.
                                                       September 28, 2019

A Survey of Transport Security Protocols
draft-ietf-taps-transport-security-09

Abstract

   This document provides a survey of commonly used or notable network
   security protocols, with a focus on how they interact and integrate
   with applications and transport protocols. Its goal is to supplement
   efforts to define and catalog transport services by describing the
   interfaces required to add security protocols. This survey is not
   limited to protocols developed within the scope or context of the
   IETF, and those included represent a superset of features a Transport
   Services system may need to support. Moreover, this document defines
   a minimal set of security features that a secure transport system
   should provide.

Table of Contents

1. Introduction

   Services and features provided by transport protocols have been
   cataloged in [RFC8095]. This document supplements that work by
   surveying commonly used and notable network security protocols, and
   identifying the services and features a Transport Services system (a
   system that provides a transport API) needs to provide in order to
   add transport security. It examines Transport Layer Security (TLS),
   Datagram Transport Layer Security (DTLS), QUIC + TLS, tcpcrypt,
   Internet Key Exchange with Encapsulating Security Protocol (IKEv2 +
   ESP), SRTP (with DTLS), WireGuard, CurveCP, and MinimalT. For each
   protocol, this document provides a brief description, the security
   features it provides, and the dependencies it has on the underlying
   transport. This is followed by defining the set of transport
   security features shared by these protocols. The document groups

these security features into a minimal set of features, which every
secure transport system should provide in addition to the transport
features described in [I-D.ietf-taps-minset], and additional optional
features, which may not be available in every secure transport
system. Finally, the document distills the application and transport
interfaces provided by the transport security protocols.

Selected protocols represent a superset of functionality and features
a Transport Services system may need to support, both internally and
externally (via an API) for applications [I-D.ietf-taps-arch].
Ubiquitous IETF protocols such as (D)TLS, as well as non-standard
protocols such as Google QUIC, are both included despite overlapping
features. As such, this survey is not limited to protocols developed
within the scope or context of the IETF. Outside of this candidate
set, protocols that do not offer new features are omitted. For
example, newer protocols such as WireGuard make unique design choices
that have important implications on applications, such as how to best
configure peer public keys and to delegate algorithm selection to the
system. In contrast, protocols such as ALTS [ALTS] are omitted since
they do not represent features deemed unique.

Authentication-only protocols such as TCP-AO [RFC5925] and IPsec AH
[RFC4302] are excluded from this survey. TCP-AO adds authenticity
protections to long-lived TCP connections, e.g., replay protection
with per-packet Message Authentication Codes. (This protocol
obsoletes TCP MD5 "signature" options specified in [RFC2385].) One
prime use case of TCP-AO is for protecting BGP connections.
Similarly, AH adds per-datagram authenticity and adds similar replay
protection. Despite these improvements, neither protocol sees
general use and both lack critical properties important for emergent
transport security protocols: confidentiality, privacy protections,
and agility. Such protocols are thus omitted from this survey.

2. Terminology

The following terms are used throughout this document to describe the
roles and interactions of transport security protocols:

o   Transport Feature: a specific end-to-end feature that the
    transport layer provides to an application. Examples include
    confidentiality, reliable delivery, ordered delivery, message-
    versus-stream orientation, etc.

o   Transport Service: a set of Transport Features, without an
    association to any given framing protocol, which provides
    functionality to an application.

o Transport Protocol: an implementation that provides one or more
   different transport services using a specific framing and header
   format on the wire. A Transport Protocol services an application.

o Application: an entity that uses a transport protocol for end-to-
   end delivery of data across the network. This may also be an
   upper layer protocol or tunnel encapsulation.

o Security Feature: a feature that a network security layer provides
   to applications. Examples include authentication, encryption, key
   generation, session resumption, and privacy. Features may be
   Mandatory or Optional for an application's implementation.
   Security Features extend the set of Transport Features described
   in [RFC8095] and provided by Transport Services implementations.

o Security Protocol: a defined network protocol that implements one
   or more security features. Security protocols may be used
   alongside transport protocols, and in combination with other
   security protocols when appropriate.

o Handshake Protocol: a protocol that enables peers to validate each
   other and to securely establish shared cryptographic context.

o Record: Framed protocol messages.

o Record Protocol: a security protocol that allows data to be
   divided into manageable blocks and protected using shared
   cryptographic context.

o Session: an ephemeral security association between applications.

o Cryptographic context: a set of cryptographic parameters,
   including but not necessarily limited to keys for encryption,
   authentication, and session resumption, enabling authorized
   parties to a session to communicate securely.

o Connection: the shared state of two or more endpoints that
   persists across messages that are transmitted between these
   endpoints. A connection is a transient participant of a session,
   and a session generally lasts between connection instances.

o Peer: an endpoint application party to a session.

o Client: the peer responsible for initiating a session.

o Server: the peer responsible for responding to a session
   initiation.

3. Security Features

   In this section, we enumerate Security Features exposed by protocols
   discussed in the remainder of this document. Protocol security (and
   privacy) properties that are unrelated to the API surface exposed by
   such protocols, such as client or server identity hiding, are not
   listed here as features.

   o Forward-secure session key establishment: Establishing
      cryptographic keys with forward-secure properties.

   o Cryptographic algorithm negotiation: Negotiating support of
      protocol algorithms, including algorithms for encryption, hashing,
      MAC (PRF), and digital signatures.

   o Session caching and management: Managing session state caches used
      for subsequent connections, with the aim of amortizing connection
      establishment costs.

   o Peer authentication: Authenticating peers using generic or
      protocol-specific mechanisms, such as certificates, raw public
      keys, pre-shared keys, or EAP methods.

   o Unilateral responder authentication: Requiring authentication for
      the responder of a connection.

   o Mutual authentication: Establishing connections in which both
      endpoints are authenticated.

   o Application authentication delegation: Delegating to applications
      out-of-band to perform peer authentication.

   o Record (channel or datagram) confidentiality and integrity:
      Encrypting and authenticating application plaintext bytes sent
      between peers over a channel or in individual datagrams.

   o Partial record confidentiality: Encrypting some portion of
      records.

   o Optional record integrity: Optionally authenticating certain
      records.

   o Record replay prevention: Detecting and defending against record
      replays, which can be due to in-network retransmissions.

   o Early data support: Transmitting application data prior to secure
      connection establishment via a handshake. For TLS, this support
      begins with TLS 1.3.

o Connection mobility: Allowing a connection to be multihomed or
   resilient across network interface or address changes, such as NAT
   rebindings that occur without an endpoint's knowledge. Mobility
   allows cryptographic key material and other state information to
   be reused in the event of a connection change.

o Application-layer feature negotiation: Securely negotiating
   application-specific functionality. Such features may be
   necessary for further application processing, such as the TLS
   parent connection protocol type via ALPN [RFC7301] or desired
   application identity via SNI [RFC6066].

o Configuration extensions: Adding protocol features via extensions
   or configuration options. TLS extensions are a primary example of
   this feature.

o Out-of-order record receipt: Processing of records received out-
   of-order.

o Source validation (cookie or puzzle based): Validating peers and
   mitigating denial-of-service (DoS) attacks via explicit proof of
   origin (cookies) or work mechanisms (puzzles).

o Length-hiding padding: Adding padding to records in order to hide
   plaintext message length and mitigate amplification attack
   vectors.

4. Transport Security Protocol Descriptions

   This section contains descriptions of security protocols currently
   used to protect data being sent over a network.

   For each protocol, we describe its provided features and dependencies
   on other protocols.

4.1. TLS

   TLS (Transport Layer Security) [RFC8446] is a common protocol used to
   establish a secure session between two endpoints. Communication over
   this session "prevents eavesdropping, tampering, and message
   forgery." TLS consists of a tightly coupled handshake and record
   protocol. The handshake protocol is used to authenticate peers,
   negotiate protocol options, such as cryptographic algorithms, and
   derive session-specific keying material. The record protocol is used
   to marshal (possibly encrypted) data from one peer to the other.
   This data may contain handshake messages or raw application data.

4.1.1. Protocol Description

   TLS is the composition of a handshake and record protocol [RFC8446].
   The record protocol is designed to marshal an arbitrary, in-order
   stream of bytes from one endpoint to the other. It handles
   segmenting, compressing (when enabled), and encrypting data into
   discrete records. When configured to use an authenticated encryption
   with associated data (AEAD) algorithm, it also handles nonce
   generation and encoding for each record. The record protocol is
   hidden from the client behind a bytestream-oriented API.

   The handshake protocol serves several purposes, including: peer
   authentication, protocol option (key exchange algorithm and
   ciphersuite) negotiation, and key derivation. Peer authentication
   may be mutual; however, commonly, only the server is authenticated.
   X.509 certificates are commonly used in this authentication step,
   though other mechanisms, such as raw public keys [RFC7250], exist.
   The client is not authenticated unless explicitly requested by the
   server.

   The handshake protocol is also extensible. It allows for a variety
   of extensions to be included by either the client or server. These
   extensions are used to specify client preferences, e.g., the
   application-layer protocol to be driven with the TLS connection
   [RFC7301], or signals to the server to aid operation, e.g., Server
   Name Indication (SNI) [RFC6066]. Various extensions also exist to
   tune the parameters of the record protocol, e.g., the maximum
   fragment length [RFC6066] and record size limit [RFC8449].

   Alerts are used to convey errors and other atypical events to the
   endpoints. There are two classes of alerts: closure and error
   alerts. A closure alert is used to signal to the other peer that the
   sender wishes to terminate the connection. The sender typically
   follows a close alert with a TCP FIN segment to close the connection.
   Error alerts are used to indicate problems with the handshake or
   individual records. Most errors are fatal and are followed by
   connection termination. However, warning alerts may be handled at
   the discretion of the implementation.

   Once a session is disconnected all session keying material must be
   destroyed, with the exception of secrets previously established
   expressly for purposes of session resumption. TLS supports stateful
   and stateless resumption. (Here, "state" refers to bookkeeping on a
   per-session basis by the server. It is assumed that the client must
   always store some state information in order to resume a session.)

4.1.2. Security Features

   o Forward-secure session key establishment.

   o Cryptographic algorithm negotiation.

   o Stateful and stateless cross-connection session resumption.

   o Session caching and management.

   o Peer authentication (Certificate, raw public key, and pre-shared
      key).

   o Unilateral responder authentication.

   o Mutual authentication.

   o Application authentication delegation.

   o Record (channel) confidentiality and integrity.

   o Record replay prevention.

   o Application-layer feature negotiation.

   o Configuration extensions.

   o Early data support (starting with TLS 1.3).

   o Optional record-layer padding (starting with TLS 1.3).

4.1.3. Protocol Dependencies

   o In-order, reliable bytestream transport.

   o (Optionally) A PKI trust store for certificate validation.

4.2. DTLS

   DTLS (Datagram Transport Layer Security) [RFC6347] is based on TLS,
   but differs in that it is designed to run over unreliable datagram
   protocols like UDP instead of TCP. DTLS modifies the protocol to
   make sure it can still provide the same security guarantees as TLS
   even without reliability from the transport. DTLS was designed to be
   as similar to TLS as possible, so this document assumes that all
   properties from TLS are carried over except where specified.

4.2.1. Protocol Description

   DTLS is modified from TLS to operate with the possibility of packet
   loss, reordering, and duplication that may occur when operating over
   UDP. To enable out-of-order delivery of application data, the DTLS
   record protocol itself has no inter-record dependencies. However, as
   the handshake requires reliability, each handshake message is
   assigned an explicit sequence number to enable retransmissions of
   lost packets and in-order processing by the receiver. Handshake
   message loss is remedied by sender retransmission after a
   configurable period in which the expected response has not yet been
   received.

   As the DTLS handshake protocol runs atop the record protocol, to
   account for long handshake messages that cannot fit within a single
   record, DTLS supports fragmentation and subsequent reconstruction of
   handshake messages across records. The receiver must reassemble
   records before processing.

   DTLS relies on unique UDP 4-tuples to identify connections, or a
   similar mechanism in other datagram transports. Since all
   application-layer data is encrypted, demultiplexing over the same
   4-tuple requires the use of a connection identifier extension
   [I-D.ietf-tls-dtls-connection-id] to permit identification of the
   correct connection-specific cryptographic context without the use of
   trial decryption. (Note that this extension is only supported in
   DTLS 1.2 and 1.3 [I-D.ietf-tls-dtls13].)

   Since datagrams can be replayed, DTLS provides optional anti-replay
   detection based on a window of acceptable sequence numbers [RFC6347].

4.2.2. Security Features

   o Record replay protection.

   o Record (datagram) confidentiality and integrity.

   o Out-of-order record receipt.

   o DoS mitigation (cookie-based).

   See also the features from TLS.

4.2.3. Protocol Dependencies

   o DTLS relies on an unreliable datagram transport.

o The DTLS record protocol explicitly encodes record lengths, so
   although it runs over a datagram transport, it does not rely on
   the transport protocol's framing beyond requiring transport-level
   reconstruction of datagrams fragmented over packets. (Note: DTLS
   1.3 short header records omit the explicit length field.)

o Uniqueness of the session within the transport flow (only one DTLS
   connection on a UDP 4-tuple, for example); or else support for the
   connection identifier extension to enable demultiplexing.

o Path MTU discovery.

o For the handshake: Reliable, in-order transport. DTLS provides
   its own reliability.

4.3. QUIC with TLS

   QUIC is a new standards-track transport protocol that runs over UDP,
   loosely based on Google's original proprietary gQUIC protocol
   [I-D.ietf-quic-transport] (See Section 4.3.4 for more details). The
   QUIC transport layer itself provides support for data confidentiality
   and integrity. This requires keys to be derived with a separate
   handshake protocol. A mapping for QUIC of TLS 1.3
   [I-D.ietf-quic-tls] has been specified to provide this handshake.

4.3.1. Protocol Description

   As QUIC relies on TLS to secure its transport functions, it creates
   specific integration points between its security and transport
   functions:

o Starting the handshake to generate keys and provide authentication
   (and providing the transport for the handshake).

o Client address validation.

o Key ready events from TLS to notify the QUIC transport.

o Exporting secrets from TLS to the QUIC transport.

   The QUIC transport layer support multiple streams over a single
   connection. QUIC implements a record protocol for TLS handshake
   messages to establish a connection. These messages are sent in
   CRYPTO frames [I-D.ietf-quic-transport] in Initial and Handshake
   packets. Initial packets are encrypted using fixed keys derived from
   the QUIC version and public packet information (Connection ID).
   Handshake packets are encrypted using TLS handshake secrets. Once
   TLS completes, QUIC uses the resulting traffic secrets to for the

QUIC connection to protect the rest of the frames. QUIC supports
0-RTT data using previously negotiated connection secrets Early data
is sent in 0-RTT packets, which may be included in the same datagram
as the Initial and Handshake packets.

4.3.2. Security Features

o DoS mitigation (cookie-based).

See also the properties of TLS.

4.3.3. Protocol Dependencies

o QUIC transport assumes an unreliable transport, e.g., UDP.

o QUIC transport relies on TLS 1.3 for key exchange, peer
   authentication, and shared secret derivation.

o For the handshake: Reliable, in-order transport. QUIC provides
   its own reliability.

4.3.4. Variant: Google QUIC

Google QUIC (gQUIC) is a UDP-based multiplexed streaming protocol
designed and deployed by Google following experience from deploying
SPDY, the proprietary predecessor to HTTP/2. gQUIC was originally
known as "QUIC": this document uses gQUIC to unambiguously
distinguish it from the standards-track IETF QUIC. The proprietary
technical forebear of IETF QUIC, gQUIC was originally designed with
tightly-integrated security and application data transport protocols.

4.4. IKEv2 with ESP

IKEv2 [RFC7296] and ESP [RFC4303] together form the modern IPsec
protocol suite that encrypts and authenticates IP packets, either for
creating tunnels (tunnel-mode) or for direct transport connections
(transport-mode). This suite of protocols separates out the key
generation protocol (IKEv2) from the transport encryption protocol
(ESP). Each protocol can be used independently, but this document
considers them together, since that is the most common pattern.

4.4.1. IKEv2 Protocol Description

IKEv2 is a control protocol that runs on UDP ports 500 or 4500 and
TCP port 4500. Its primary goal is to generate keys for Security
Associations (SAs). An SA contains shared (cryptographic)
information used for establishing other SAs or keying ESP; See
Section 4.4.2. IKEv2 first uses a Diffie-Hellman key exchange to

generate keys for the "IKE SA", which is a set of keys used to
encrypt further IKEv2 messages. IKE then performs a phase of
authentication in which both peers present blobs signed by a shared
secret or private key that authenticates the entire IKE exchange and
the IKE identities. IKE then derives further sets of keys on demand,
which together with traffic policies are referred to as the "Child
SA". These Child SA keys are used by ESP.

IKEv2 negotiates which protocols are acceptable to each peer for both
the IKE and Child SAs using "Proposals". Each proposal specifies an
encryption and authentication algorithm, or an AEAD algorithm, a
Diffie-Hellman group, and (for IKE SAs only) a pseudorandom function
algorithm. Each peer may support multiple proposals, and the most
preferred mutually supported proposal is chosen during the handshake.

The authentication phase of IKEv2 may use Shared Secrets,
Certificates, Digital Signatures, or an EAP (Extensible
Authentication Protocol) method. At a minimum, IKEv2 takes two round
trips to set up both an IKE SA and a Child SA. If EAP is used, this
exchange may be expanded.

Any SA used by IKEv2 can be re-keyed before expiration, which is
usually based either on time or number of bytes encrypted.

There is an extension to IKEv2 that allows session resumption
[RFC5723].

MOBIKE is a Mobility and Multihoming extension to IKEv2 that allows a
set of Security Associations to migrate over different outer IP
addresses and interfaces [RFC4555].

When UDP is not available or well-supported on a network, IKEv2 may
be encapsulated in TCP [RFC8229].

4.4.2. ESP Protocol Description

ESP is a protocol that encrypts and authenticates IPv4 and IPv6
packets. The keys used for both encryption and authentication can be
derived from an IKEv2 exchange. ESP Security Associations come as
pairs, one for each direction between two peers. Each SA is
identified by a Security Parameter Index (SPI), which is marked on
each encrypted ESP packet.

ESP packets include the SPI, a sequence number, an optional
Initialization Vector (IV), payload data, padding, a length and next
header field, and an Integrity Check Value.

From [RFC4303], "ESP is used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and limited traffic flow confidentiality."

Since ESP operates on IP packets, it is not directly tied to the transport protocols it encrypts. This means it requires little or no change from transports in order to provide security.

ESP packets may be sent directly over IP, but where network conditions warrant (e.g., when a NAT is present or when a firewall blocks such packets) they may be encapsulated in UDP [RFC3948] or TCP [RFC8229].

4.4.3. IKEv2 Security Features

   o Forward-secure session key establishment.

   o Cryptographic algorithm negotiation.

   o Peer authentication (certificate, raw public key, pre-shared key, and EAP).

   o Unilateral responder authentication.

   o Mutual authentication.

   o Record (datagram) confidentiality and integrity.

   o Session resumption.

   o Connection mobility.

   o DoS mitigation (cookie-based).

4.4.4. ESP Security Features

   o Record confidentiality and integrity.

   o Record replay protection.

4.4.5. IKEv2 Protocol Dependencies

   o Availability of UDP to negotiate, or implementation support for TCP-encapsulation.

   o Some EAP authentication types require accessing a hardware device,
      such as a SIM card; or interacting with a user, such as password
      prompting.

4.4.6. ESP Protocol Dependencies

   o Since ESP is below transport protocols, it does not have any
      dependencies on the transports themselves, other than on UDP or
      TCP where encapsulation is employed.

4.5. Secure RTP (with DTLS)

   Secure RTP (SRTP) is a profile for RTP that provides confidentiality,
   message authentication, and replay protection for RTP data packets
   and RTP control protocol (RTCP) packets [RFC3711].

4.5.1. Protocol description

   SRTP adds confidentiality and optional integrity protection to RTP
   data packets, and adds confidentially and mandatory integrity
   protection to RTCP packets. For RTP data packets, this is done by
   encrypting the payload section of the packet and optionally appending
   an authentication tag (MAC) as a packet trailer, with the RTP header
   authenticated but not encrypted (the RTP header was left unencrypted
   to enable RTP header compression [RFC2508] [RFC3545]). For RTCP
   packets, the first packet in the compound RTCP packet is partially
   encrypted, leaving the first eight octets of the header as clear-text
   to allow identification of the packet as RTCP, while the remainder of
   the compound packet is fully encrypted. The entire RTCP packet is
   then authenticated by appending a MAC as packet trailer.

   Packets are encrypted using session keys, which are ultimately
   derived from a master key and an additional master salt and session
   salt. SRTP packets carry a 2-byte sequence number to partially
   identify the unique packet index. SRTP peers maintain a separate
   roll-over counter (ROC) for RTP data packets that is incremented
   whenever the sequence number wraps. The sequence number and ROC
   together determine the packet index. RTCP packets have a similar,
   yet differently named, field called the RTCP index which serves the
   same purpose.

   Numerous encryption modes are supported. For popular modes of
   operation, e.g., AES-CTR, the (unique) initialization vector (IV)
   used for each encryption mode is a function of the RTP SSRC
   (synchronization source), packet index, and session "salting key".

SRTP offers replay detection by keeping a replay list of already seen
and processed packet indices. If a packet arrives with an index that
matches one in the replay list, it is silently discarded.

DTLS [RFC5764] is commonly used to perform mutual authentication and
key agreement for SRTP [RFC5763]. Peers use DTLS to perform mutual
certificate-based authentication on the media path, and to generate
the SRTP master key. Peer certificates can be issued and signed by a
certificate authority. Alternatively, certificates used in the DTLS
exchange can be self-signed. If they are self-signed, certificate
fingerprints are included in the signaling exchange (e.g., in SIP or
WebRTC), and used to bind the DTLS key exchange in the media plane to
the signaling plane. The combination of a mutually authenticated
DTLS key exchange on the media path and a fingerprint sent in the
signaling channel protects against active attacks on the media,
provided the signaling can be trusted. Signaling needs to be
protected as described in, for example, SIP [RFC3261] Authenticated
Identity Management [RFC8224] or the WebRTC security architecture
[I-D.ietf-rtcweb-security-arch], to provide complete system security.

4.5.2. Security Features

   o Forward-secure session key establishment.

   o Cryptographic algorithm negotiation.

   o Mutual authentication.

   o Partial datagram confidentiality. (Packet headers are not
      encrypted.)

   o Optional authentication of data packets.

   o Mandatory authentication of control packets.

   o Out-of-order record receipt.

4.5.3. Protocol Dependencies

   o Secure RTP can run over UDP or TCP.

   o External key derivation and management protocol, e.g., DTLS
      [RFC5763].

   o External identity management protocol, e.g., SIP Authenticated
      Identity Management [RFC8224], WebRTC Security Architecture
      [I-D.ietf-rtcweb-security-arch].

4.5.4. Variant: ZRTP for Media Path Key Agreement

   ZRTP [RFC6189] is an alternative key agreement protocol for SRTP. It
   uses standard SRTP to protect RTP data packets and RTCP packets, but
   provides alternative key agreement and identity management protocols.

   Key agreement is performed using a Diffie-Hellman key exchange that
   runs on the media path. This generates a shared secret that is then
   used to generate the master key and salt for SRTP.

   ZRTP does not rely on a PKI or external identity management system.
   Rather, it uses an ephemeral Diffie-Hellman key exchange with hash
   commitment to allow detection of man-in-the-middle attacks. This
   requires endpoints to display a short authentication string that the
   users must read and verbally compare to validate the hashes and
   ensure security. Endpoints cache some key material after the first
   call to use in subsequent calls; this is mixed in with the Diffie-
   Hellman shared secret, so the short authentication string need only
   be checked once for a given user. This gives key continuity
   properties analogous to the secure shell (ssh) [RFC4253].

4.6. tcpcrypt

   Tcpcrypt [RFC8548] is a lightweight extension to the TCP protocol for
   opportunistic encryption. Applications may use tcpcrypt's unique
   session ID for further application-level authentication. Absent this
   authentication, tcpcrypt is vulnerable to active attacks.

4.6.1. Protocol Description

   Tcpcrypt extends TCP to enable opportunistic encryption between the
   two ends of a TCP connection [RFC8548]. It is a family of TCP
   encryption protocols (TEP), distinguished by key exchange algorithm.
   The use of a TEP is negotiated with a TCP option during the initial
   TCP handshake via the mechanism described by TCP Encryption
   Negotiation Option (ENO) [RFC8547]. In the case of initial session
   establishment, once a tcpcrypt TEP has been negotiated the key
   exchange occurs within the data segments of the first few packets
   exchanged after the handshake completes. The initiator of a
   connection sends a list of supported AEAD algorithms, a random nonce,
   and an ephemeral public key share. The responder typically chooses a
   mutually-supported AEAD algorithm and replies with this choice, its
   own nonce, and ephemeral key share. An initial shared secret is
   derived from the ENO handshake, the tcpcrypt handshake, and the
   initial keying material resulting from the key exchange. The traffic
   encryption keys on the initial connection are derived from the shared
   secret. Connections can be re-keyed before the natural AEAD limit
   for a single set of traffic encryption keys is reached.

Each tcpcrypt session is associated with a ladder of resumption IDs, each derived from the respective entry in a ladder of shared secrets. These resumption IDs can be used to negotiate a stateful resumption of the session in a subsequent connection, resulting in use of a new shared secret and traffic encryption keys without requiring a new key exchange. Willingness to resume a session is signaled via the ENO option during the TCP handshake. Given the length constraints imposed by TCP options, unlike stateless resumption mechanisms (such as that provided by session tickets in TLS) resumption in tcpcrypt requires the maintenance of state on the server, and so successful resumption across a pool of servers implies shared state.

Owing to middlebox ossification issues, tcpcrypt only protects the payload portion of a TCP packet. It does not encrypt any header information, such as the TCP sequence number.

4.6.2. Security Features

   o Forward-secure session key establishment.

   o Record (channel) confidentiality and integrity.

   o Stateful cross-connection session resumption.

   o Session caching and management.

   o Application authentication delegation.

4.6.3. Protocol Dependencies

   o TCP for in-order, reliable transport.

   o TCP Encryption Negotiation Option (ENO).

4.7. WireGuard

   WireGuard is a layer 3 protocol designed as an alternative to IPsec [WireGuard] for certain use cases. It uses UDP to encapsulate IP datagrams between peers. Unlike most transport security protocols, which rely on PKI for peer authentication, WireGuard authenticates peers using pre-shared public keys delivered out-of-band, each of which is bound to one or more IP addresses. Moreover, as a protocol suited for VPNs, WireGuard offers no extensibility, negotiation, or cryptographic agility.

4.7.1. Protocol description

   WireGuard is a simple VPN protocol that binds a pre-shared public key
   to one or more IP addresses. Users configure WireGuard by
   associating peer public keys with IP addresses. These mappings are
   stored in a CryptoKey Routing Table. (See Section 2 of [WireGuard]
   for more details and sample configurations.) These keys are used
   upon WireGuard packet transmission and reception. For example, upon
   receipt of a Handshake Initiation message, receivers use the static
   public key in their CryptoKey routing table to perform necessary
   cryptographic computations.

   WireGuard builds on Noise [Noise] for 1-RTT key exchange with
   identity hiding. The handshake hides peer identities as per the
   SIGMA construction [SIGMA]. As a consequence of using Noise,
   WireGuard comes with a fixed set of cryptographic algorithms:

   o x25519 [Curve25519] and HKDF [RFC5869] for ECDH and key
      derivation.

   o ChaCha20+Poly1305 [RFC8439] for packet authenticated encryption.

   o BLAKE2s [BLAKE2] for hashing.

   There is no cryptographic agility. If weaknesses are found in any of
   these algorithms, new message types using new algorithms must be
   introduced.

   If a WireGuard receiver is under heavy load and cannot process a
   packet, e.g., cannot spare CPU cycles for expensive public key
   cryptographic operations, it can reply with a cookie similar to DTLS
   and IKEv2. This cookie only proves IP address ownership. Any rate
   limiting scheme can be applied to packets coming from non-spoofed
   addresses.

4.7.2. Security Features

   o Forward-secure session key establishment.

   o Peer authentication (public-key and PSK).

   o Mutual authentication.

   o Record replay prevention (Stateful, timestamp-based).

   o Connection mobility.

   o DoS mitigation (cookie-based).

4.7.3. Protocol Dependencies

   o Datagram transport.

   o Out-of-band key distribution and management.

4.8. CurveCP

   CurveCP [CurveCP] is a UDP-based transport security protocol from
   Daniel J. Bernstein. Unlike other transport security protocols, it
   is based entirely upon highly efficient public key algorithms. This
   removes many pitfalls associated with nonce reuse and key
   synchronization.

4.8.1. Protocol Description

   CurveCP is a UDP-based transport security protocol. It is built on
   three principal features: exclusive use of public key authenticated
   encryption of packets, server-chosen cookies to prohibit memory and
   computation DoS at the server, and connection mobility with a client-
   chosen ephemeral identifier.

   There are two rounds in CurveCP. In the first round, the client
   sends its first initialization packet to the server, carrying its
   (possibly fresh) ephemeral public key C', with zero-padding encrypted
   under the server's long-term public key. The server replies with a
   cookie and its own ephemeral key S' and a cookie that is to be used
   by the client. Upon receipt, the client then generates its second
   initialization packet carrying: the ephemeral key C', cookie, and an
   encryption of C', the server's domain name, and, optionally, some
   message data. The server verifies the cookie and the encrypted
   payload and, if valid, proceeds to send data in return. At this
   point, the connection is established and the two parties can
   communicate.

   The use of public-key encryption and authentication, or "boxing",
   simplifies problems that come with symmetric key management and nonce
   synchronization. For example, it allows the sender of a message to
   be in complete control of each message's nonce. It does not require
   either end to share secret keying material. Furthermore, it allows
   connections (or sessions) to be associated with unique ephemeral
   public keys as a mechanism for enabling forward secrecy given the
   risk of long-term private key compromise.

   The client and server do not perform a standard key exchange.
   Instead, in the initial exchange of packets, each party provides its
   own ephemeral key to the other end. The client can choose a new
   ephemeral key for every new connection. However, the server must

rotate these keys on a slower basis. Otherwise, it would be trivial
for an attacker to force the server to create and store ephemeral
keys with a fake client initialization packet.

Servers use cookies for source validation. After receiving a
client's initial packet, encrypted under the server's long-term
public key, a server generates and returns a stateless cookie that
must be echoed back in the client's following message. This cookie
is encrypted under the client's ephemeral public key. This stateless
technique prevents attackers from hijacking client initialization
packets to obtain cookie values to flood clients. (A client would
detect the duplicate cookies and reject the flooded packets.)
Similarly, replaying the client's second packet, carrying the cookie,
will be detected by the server.

CurveCP supports client authentication by allowing clients to send
their long-term public keys in the second initialization packet. A
server can verify this public key and, if untrusted, drop the
connection and subsequent data.

Unlike some other protocols, CurveCP data packets leave only the
ephemeral public key, connection ID, and per-message nonce in the
clear. All other data is encrypted.

4.8.2. Protocol Features

   o Datagram confidentiality and integrity (via public key
      encryption).

   o Peer authentication (public-key).

   o Unilateral responder authentication.

   o Mutual authentication.

   o Connection mobility (based on a client-chosen ephemeral
      identifier).

   o Optional length-hiding and anti-amplification padding.

   o Source validation (cookie-based)

4.8.3. Protocol Dependencies

   o An unreliable transport protocol such as UDP.

4.9. MinimalT

   MinimalT is a UDP-based transport security protocol designed to offer
   confidentiality, mutual authentication, DoS prevention, and
   connection mobility [MinimalT]. One major goal of the protocol is to
   leverage existing protocols to obtain server-side configuration
   information used to more quickly bootstrap a connection. MinimalT
   uses a variant of TCP's congestion control algorithm.

4.9.1. Protocol Description

   MinimalT is a secure transport protocol built on top of a widespread
   directory service. Clients and servers interact with local directory
   services to (a) resolve server information and (b) publish ephemeral
   state information, respectively. Clients connect to a local resolver
   once at boot time. Through this resolver they recover the IP
   address(es) and public key(s) of each server to which they want to
   connect.

   Connections are instances of user-authenticated, mobile sessions
   between two endpoints. Connections run within tunnels between hosts.
   A tunnel is a server-authenticated container that multiplexes
   multiple connections between the same hosts. All connections in a
   tunnel share the same transport state machine and encryption. Each
   tunnel has a dedicated control connection used to configure and
   manage the tunnel over time. Moreover, since tunnels are independent
   of the network address information, they may be reused as both ends
   of the tunnel move about the network. This does however imply that
   connection establishment and packet encryption mechanisms are
   coupled.

   Before a client connects to a remote service, it must first establish
   a tunnel to the host providing or offering the service. Tunnels are
   established in 1-RTT using an ephemeral key obtained from the
   directory service. Tunnel initiators provide their own ephemeral key
   and, optionally, a DoS puzzle solution such that the recipient
   (server) can verify the authenticity of the request and derive a
   shared secret. Within a tunnel, new connections to services may be
   established.

   Additional (orthogonal) transport features include: connection
   multiplexing between hosts across shared tunnels, and congestion
   control state is shared across connections between the same host
   pairs.

4.9.2. Protocol Features

   o Record or datagram confidentiality and integrity.

   o Forward-secure session key establishment.

   o Peer authentication (public-key).

   o Unilateral responder authentication.

   o DoS mitigation (puzzle-based).

   o Out-of-order receipt record.

   o Connection mobility (based on tunnel identifiers).

4.9.3. Protocol Dependencies

   o An unreliable transport protocol such as UDP.

   o A DNS-like resolution service to obtain location information (an
      IP address) and ephemeral keys.

   o A PKI trust store for certificate validation.

4.10. OpenVPN

   OpenVPN [OpenVPN] is a commonly used protocol designed as an
   alternative to IPsec. A major goal of this protocol is to provide a
   VPN that is simple to configure and works over a variety of
   transports. OpenVPN encapsulates either IP packets or Ethernet
   frames within a secure tunnel and can run over UDP or TCP.

4.10.1. Protocol Description

   OpenVPN facilitates authentication using either a pre-shared static
   key or using X.509 certificates and TLS. In pre-shared key mode,
   OpenVPN derives keys for encryption and authentication directly from
   one or multiple symmetric keys. In TLS mode, OpenVPN encapsulates a
   TLS handshake, in which both peers must present a certificate for
   authentication. After the handshake, both sides contribute random
   source material to derive keys for encryption and authentication
   using the TLS pseudo random function (PRF). OpenVPN provides the
   possibility to authenticate and encrypt the TLS handshake itself
   using a pre-shared key or passphrase. Furthermore, it supports re-
   keying using TLS.

After authentication and key exchange, OpenVPN encrypts payload data, i.e., IP packets or Ethernet frames, and authenticates the payload using HMAC. Applications can select an arbitrary encryption algorithm (cipher) and key size, as well hash function for HMAC. The default cipher and hash functions are AES-GCM and SHA1, respectively. Recent versions of the protocol support cipher negotiation.

OpenVPN can run over TCP or UDP. When running over UDP, OpenVPN provides a simple reliability layer for control packets such as the TLS handshake and key exchange. It assigns sequence numbers to packets, acknowledges packets it receives, and retransmits packets it deems lost. Similar to DTLS, this reliability layer is not used for data packets, which prevents the problem of two reliability mechanisms being encapsulated within each other. When running over TCP, OpenVPN includes the packet length in the header, which allows the peer to deframe the TCP stream into messages.

For replay protection, OpenVPN assigns an identifier to each outgoing packet, which is unique for the packet and the currently used key. In pre-shared key mode or with a CFB or OFB mode cipher, OpenVPN combines a timestamp with an incrementing sequence number into a 64-bit identifier. In TLS mode with CBC cipher mode, OpenVPN omits the timestamp, so identifiers are only 32-bit. This is sufficient since OpenVPN can guarantee the uniqueness of this identifier for each key, as it can trigger re-keying if needed.

OpenVPN supports connection mobility by allowing a peer to change its IP address during an ongoing session. When configured accordingly, a host will accept authenticated packets for a session from any IP address.

4.10.2. Protocol Features

   o Peer authentication using certificates or pre-shared key.

   o Mandatory mutual authentication.

   o Connection mobility.

   o Out-of-order record receipt.

   o Length-hiding padding.

   See also the properties of TLS.

4.10.3. Protocol Dependencies

   o For control packets such as handshake and key exchange: Reliable,
      in-order transport. Reliability is provided either by TCP, or by
      OpenVPN's own reliability layer when using UDP.

5. Security Features and Application Dependencies

   There exists a common set of features shared across the transport
   protocols surveyed in this document. Mandatory features constitute a
   baseline of functionality that an application may assume for any
   Transport Services implementation. They were selected on the basis
   that they are either (a) required for any secure transport protocol
   or (b) nearly ubiquitous amongst common secure transport protocols.

   Optional features by contrast may vary from implementation to
   implementation, and so an application cannot simply assume they are
   available. Applications learn of and use optional features by
   querying for their presence and support. Optional features may not
   be implemented, or may be disabled if their presence impacts
   transport services or if a necessary transport service or application
   dependency is unavailable.

   In this context, an application dependency is an aspect of the
   security feature which can be exposed to the application. An
   application dependency may be required for the security feature to
   function, or it may provide additional information and control to the
   application. For example, an application may need to provide
   information such as keying material or authentication credentials, or
   it may want to restrict which cryptographic algorithms to allow for
   negotiation.

5.1. Mandatory Features

   Mandatory features must be supported regardless of transport and
   application services available. Note that not all mandatory features
   are provided by each surveyed protocol above. For example, tcpcrypt
   does not provide responder authentication and CurveCP does not
   provide forward-secure session key establishment.

   o Record or datagram confidentiality and integrity.

      * Application dependency: None.

   o Forward-secure session key establishment.

      * Application dependency: None.

o Unilateral responder authentication.

   * (Optional) Application dependency: Application-provided trust
      information. System trust stores may also be used to
      authenticate responders.

5.2. Optional Features

   In this section we list optional features along with their necessary
   application dependencies, if any.

o Pre-shared key support (PSK):

   * Application dependency: Application provisioning and
      distribution of pre-shared keys.

o Mutual authentication (MA):

   * Application dependency: Mutual authentication credentials
      required.

o Cryptographic algorithm negotiation (AN):

   * Application dependency: Application awareness of supported or
      desired algorithms.

o Application authentication delegation (AD):

   * Application dependency: Application opt-in and policy for
      endpoint authentication.

o DoS mitigation (DM):

   * Application dependency: None.

o Connection mobility (CM):

   * Application dependency: None.

o Source validation (SV):

   * Application dependency: None.

o Application-layer feature negotiation (AFN):

   * Application dependency: Specification of application-layer
      features or functionality.

o Configuration extensions (CX):

   * Application dependency: Specification of application-specific
      extensions.

o Session caching and management (SC):

   * Application dependency: None.

o Length-hiding padding (LHP): (Optional) Application dependency:
   Knowledge of desired padding policies. Some protocols, such as
   IKE, can negotiate application-agnostic padding policies.

o Early data support (ED):

   * Application dependency: Anti-replay protections or hints of
      data idempotency.

o Record replay prevention (RP):

   * Application dependency: None.

o Out-of-order receipt record (OO):

   * Application dependency: None.

5.3. Optional Feature Availability

   The following table lists the availability of the above-listed
   optional features in each of the analyzed protocols. "Mandatory"
   indicates that the feature is intrinsic to the protocol and cannot be
   disabled. "Supported" indicates that the feature is optionally
   provided natively or through a (standardized, where applicable)
   extension.

Protocol	PSK	AN	AD	MA	DM	CM	SV	AFN	CX	SC	LHP	ED	RP	OO
TLS	S	S	S	S	S	U*	M	S	S	S	S	S	U	U
DTLS	S	S	S	S	S	S	M	S	S	S	S	U	M	M
QUIC	S	S	S	S	S	S	M	S	S	S	S	S	M	M
IKEv2+ESP	S	S	S	M	S	S	M	S	S	S	S	U	M	M
SRTP+DTLS	S	S	S	S	S	U	M	S	S	S	U	U	M	M
tcpcrypt	U	S	M	U	U**	U*	M	U	U	S	U	U	U	U
WireGuard	S	U	S	M	S	U	M	U	U	U	S+	U	M	M
MinimalT	U	U	U	M	S	M	M	U	U	U	S	U	U	U
CurveCP	U	U	U	S	S	M	M	U	U	U	S	U	M	M

M=Mandatory S=Supported but not required U=Unsupported *=On TCP; MPTCP would provide this ability **=TCP provides SYN cookies natively, but these are not cryptographically strong +=For transport packets only

6. Transport Security Protocol Interfaces

   This section describes the interface surface exposed by the security
   protocols described above. Note that not all protocols support each
   interface. We partition these interfaces into pre-connection
   (configuration), connection, and post-connection interfaces,
   following conventions in [I-D.ietf-taps-interface] and
   [I-D.ietf-taps-arch].

6.1. Pre-Connection Interfaces

   Configuration interfaces are used to configure the security protocols
   before a handshake begins or the keys are negotiated.

   o Identities and Private Keys The application can provide its
      identities (certificates) and private keys, or mechanisms to
      access these, to the security protocol to use during handshakes.
      Protocols: TLS, DTLS, QUIC + TLS, MinimalT, CurveCP, IKEv2,
      WireGuard, SRTP

   o Supported Algorithms (Key Exchange, Signatures, and Ciphersuites)
      The application can choose the algorithms that are supported for
      key exchange, signatures, and ciphersuites. Protocols: TLS, DTLS,
      QUIC + TLS, MinimalT, tcpcrypt, IKEv2, SRTP

   o Extensions (Application-Layer Protocol Negotiation): The
      application enables or configures extensions that are to be
      negotiated by the security protocol, such as ALPN [RFC7301].
      Protocols: TLS, DTLS, QUIC + TLS

   o Session Cache Management The application provides the ability to
      save and retrieve session state (such as tickets, keying material,
      and server parameters) that may be used to resume the security
      session. Protocols: TLS, DTLS, QUIC + TLS, MinimalT

   o Authentication Delegation The application provides access to a
      separate module that will provide authentication, using EAP for
      example. Protocols: IKEv2, SRTP

   o Pre-Shared Key Import Either the handshake protocol or the
      application directly can supply pre-shared keys for the record
      protocol use for encryption/decryption and authentication. If the
      application can supply keys directly, this is considered explicit
      import; if the handshake protocol traditionally provides the keys
      directly, it is considered direct import; if the keys can only be
      shared by the handshake, they are considered non-importable.

      * Explicit import: QUIC, ESP

       *   Direct import: TLS, DTLS, MinimalT, tcpcrypt, WireGuard

       *   Non-importable: CurveCP

6.2. Connection Interfaces

   o Identity Validation During a handshake, the security protocol will
      conduct identity validation of the peer. This can call into the
      application to offload validation. Protocols: All (TLS, DTLS,
      QUIC + TLS, MinimalT, CurveCP, IKEv2, WireGuard, SRTP (DTLS))

   o Source Address Validation The handshake protocol may delegate
      validation of the remote peer that has sent data to the transport
      protocol or application. This involves sending a cookie exchange
      to avoid DoS attacks. Protocols: QUIC + TLS, DTLS, WireGuard

6.3. Post-Connection Interfaces

   o Connection Termination The security protocol may be instructed to
      tear down its connection and session information. This is needed
      by some protocols to prevent application data truncation attacks.
      Protocols: TLS, DTLS, QUIC, tcpcrypt, IKEv2, MinimalT

   o Key Update The handshake protocol may be instructed to update its
      keying material, either by the application directly or by the
      record protocol sending a key expiration event. Protocols: TLS,
      DTLS, QUIC, tcpcrypt, IKEv2, MinimalT

   o Pre-Shared Key Export The handshake protocol will generate one or
      more keys to be used for record encryption/decryption and
      authentication. These may be explicitly exportable to the
      application, traditionally limited to direct export to the record
      protocol, or inherently non-exportable because the keys must be
      used directly in conjunction with the record protocol.

       *   Explicit export: TLS (for QUIC), tcpcrypt, IKEv2, DTLS (for
           SRTP)

       *   Direct export: TLS, DTLS, MinimalT

       *   Non-exportable: CurveCP

   o Key Expiration The record protocol can signal that its keys are
      expiring due to reaching a time-based deadline, or a use-based
      deadline (number of bytes that have been encrypted with the key).
      This interaction is often limited to signaling between the record
      layer and the handshake layer. Protocols: ESP ((Editor's note:
      One may consider TLS/DTLS to also have this interface))

o Mobility Events The record protocol can be signaled that it is
   being migrated to another transport or interface due to connection
   mobility, which may reset address and state validation and induce
   state changes such as use of a new Connection Identifier (CID).
   Protocols: QUIC, MinimalT, CurveCP, ESP, WireGuard (roaming)

7. IANA Considerations

   This document has no request to IANA.

8. Security Considerations

   This document summarizes existing transport security protocols and
   their interfaces. It does not propose changes to or recommend usage
   of reference protocols. Moreover, no claims of security and privacy
   properties beyond those guaranteed by the protocols discussed are
   made. For example, metadata leakage via timing side channels and
   traffic analysis may compromise any protocol discussed in this
   survey. Applications using Security Interfaces should take such
   limitations into consideration when using a particular protocol
   implementation.

9. Privacy Considerations

   Analysis of how features improve or degrade privacy is intentionally
   omitted from this survey. All security protocols surveyed generally
   improve privacy by reducing information leakage via encryption.
   However, varying amounts of metadata remain in the clear across each
   protocol. For example, client and server certificates are sent in
   cleartext in TLS 1.2 [RFC5246], whereas they are encrypted in TLS 1.3
   [RFC8446]. A survey of privacy features, or lack thereof, for
   various security protocols could be addressed in a separate document.

10. Acknowledgments

   The authors would like to thank Bob Bradley, Frederic Jacobs, Mirja
   Kuehlewind, Yannick Sierra, Brian Trammell, and Magnus Westerlund for
   their input and feedback on this draft.

11. Informative References

   [ALTS]   Ghali, C., Stubblefield, A., Knapp, E., Li, J., Schmidt,
              B., and J. Boeuf, "Application Layer Transport Security",
              <https://cloud.google.com/security/encryption-in-transit/
              application-layer-transport-security/>.

   [BLAKE2]   Aumasson, J., Neves, S., Wilcox-O'Hearn, Z., and C.
               Winnerlein, "BLAKE2 -- simpler, smaller, fast as MD5",
               <https://blake2.net/blake2.pdf>.

   [Curve25519]
               Bernstein, D., "Curve25519 - new Diffie-Hellman speed
               records", <https://cr.yp.to/ecdh/curve25519-20060209.pdf>.

   [CurveCP]   Bernstein, D., "CurveCP -- Usable security for the
               Internet", <http://curvecp.org>.

   [I-D.ietf-quic-tls]
               Thomson, M. and S. Turner, "Using TLS to Secure QUIC",
               draft-ietf-quic-tls-23 (work in progress), September 2019.

   [I-D.ietf-quic-transport]
               Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed
               and Secure Transport", draft-ietf-quic-transport-23 (work
               in progress), September 2019.

   [I-D.ietf-rtcweb-security-arch]
               Rescorla, E., "WebRTC Security Architecture", draft-ietf-
               rtcweb-security-arch-20 (work in progress), July 2019.

   [I-D.ietf-taps-arch]
               Pauly, T., Trammell, B., Brunstrom, A., Fairhurst, G.,
               Perkins, C., Tiesel, P., and C. Wood, "An Architecture for
               Transport Services", draft-ietf-taps-arch-04 (work in
               progress), July 2019.

   [I-D.ietf-taps-interface]
               Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G.,
               Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T.
               Pauly, "An Abstract Application Layer Interface to
               Transport Services", draft-ietf-taps-interface-04 (work in
               progress), July 2019.

   [I-D.ietf-taps-minset]
               Welzl, M. and S. Gjessing, "A Minimal Set of Transport
               Services for End Systems", draft-ietf-taps-minset-11 (work
               in progress), September 2018.

   [I-D.ietf-tls-dtls-connection-id]
               Rescorla, E., Tschofenig, H., and T. Fossati, "Connection
               Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-
               id-06 (work in progress), July 2019.

   [I-D.ietf-tls-dtls13]
              Rescorla, E., Tschofenig, H., and N. Modadugu, "The
              Datagram Transport Layer Security (DTLS) Protocol Version
              1.3", draft-ietf-tls-dtls13-32 (work in progress), July
              2019.

   [MinimalT]
              Petullo, W., Zhang, X., Solworth, J., Bernstein, D., and
              T. Lange, "MinimaLT -- Minimal-latency Networking Through
              Better Security",
              <http://dl.acm.org/citation.cfm?id=2516737>.

   [Noise]   Perrin, T., "The Noise Protocol Framework",
              <http://noiseprotocol.org/noise.pdf>.

   [OpenVPN] "OpenVPN cryptographic layer", <https://openvpn.net/
              community-resources/openvpn-cryptographic-layer/>.

   [RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5
              Signature Option", RFC 2385, DOI 10.17487/RFC2385, August
              1998, <https://www.rfc-editor.org/info/rfc2385>.

   [RFC2508] Casner, S. and V. Jacobson, "Compressing IP/UDP/RTP
              Headers for Low-Speed Serial Links", RFC 2508,
              DOI 10.17487/RFC2508, February 1999,
              <https://www.rfc-editor.org/info/rfc2508>.

   [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
              A., Peterson, J., Sparks, R., Handley, M., and E.
              Schooler, "SIP: Session Initiation Protocol", RFC 3261,
              DOI 10.17487/RFC3261, June 2002,
              <https://www.rfc-editor.org/info/rfc3261>.

   [RFC3545] Koren, T., Casner, S., Geevarghese, J., Thompson, B., and
              P. Ruddy, "Enhanced Compressed RTP (CRTP) for Links with
              High Delay, Packet Loss and Reordering", RFC 3545,
              DOI 10.17487/RFC3545, July 2003,
              <https://www.rfc-editor.org/info/rfc3545>.

   [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K.
              Norrman, "The Secure Real-time Transport Protocol (SRTP)",
              RFC 3711, DOI 10.17487/RFC3711, March 2004,
              <https://www.rfc-editor.org/info/rfc3711>.

   [RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M.
              Stenberg, "UDP Encapsulation of IPsec ESP Packets",
              RFC 3948, DOI 10.17487/RFC3948, January 2005,
              <https://www.rfc-editor.org/info/rfc3948>.

   [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH)
              Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253,
              January 2006, <https://www.rfc-editor.org/info/rfc4253>.

   [RFC4302] Kent, S., "IP Authentication Header", RFC 4302,
              DOI 10.17487/RFC4302, December 2005,
              <https://www.rfc-editor.org/info/rfc4302>.

   [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)",
              RFC 4303, DOI 10.17487/RFC4303, December 2005,
              <https://www.rfc-editor.org/info/rfc4303>.

   [RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol
              (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006,
              <https://www.rfc-editor.org/info/rfc4555>.

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

   [RFC5723] Sheffer, Y. and H. Tschofenig, "Internet Key Exchange
              Protocol Version 2 (IKEv2) Session Resumption", RFC 5723,
              DOI 10.17487/RFC5723, January 2010,
              <https://www.rfc-editor.org/info/rfc5723>.

   [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework
              for Establishing a Secure Real-time Transport Protocol
              (SRTP) Security Context Using Datagram Transport Layer
              Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May
              2010, <https://www.rfc-editor.org/info/rfc5763>.

   [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer
              Security (DTLS) Extension to Establish Keys for the Secure
              Real-time Transport Protocol (SRTP)", RFC 5764,
              DOI 10.17487/RFC5764, May 2010,
              <https://www.rfc-editor.org/info/rfc5764>.

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

   [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP
              Authentication Option", RFC 5925, DOI 10.17487/RFC5925,
              June 2010, <https://www.rfc-editor.org/info/rfc5925>.

   [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS)
              Extensions: Extension Definitions", RFC 6066,
              DOI 10.17487/RFC6066, January 2011,
              <https://www.rfc-editor.org/info/rfc6066>.

   [RFC6189] Zimmermann, P., Johnston, A., Ed., and J. Callas, "ZRTP:
              Media Path Key Agreement for Unicast Secure RTP",
              RFC 6189, DOI 10.17487/RFC6189, April 2011,
              <https://www.rfc-editor.org/info/rfc6189>.

   [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
              Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347,
              January 2012, <https://www.rfc-editor.org/info/rfc6347>.

   [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J.,
              Weiler, S., and T. Kivinen, "Using Raw Public Keys in
              Transport Layer Security (TLS) and Datagram Transport
              Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250,
              June 2014, <https://www.rfc-editor.org/info/rfc7250>.

   [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T.
              Kivinen, "Internet Key Exchange Protocol Version 2
              (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October
              2014, <https://www.rfc-editor.org/info/rfc7296>.

   [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan,
              "Transport Layer Security (TLS) Application-Layer Protocol
              Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301,
              July 2014, <https://www.rfc-editor.org/info/rfc7301>.

   [RFC8095] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind,
              Ed., "Services Provided by IETF Transport Protocols and
              Congestion Control Mechanisms", RFC 8095,
              DOI 10.17487/RFC8095, March 2017,
              <https://www.rfc-editor.org/info/rfc8095>.

   [RFC8224] Peterson, J., Jennings, C., Rescorla, E., and C. Wendt,
              "Authenticated Identity Management in the Session
              Initiation Protocol (SIP)", RFC 8224,
              DOI 10.17487/RFC8224, February 2018,
              <https://www.rfc-editor.org/info/rfc8224>.

   [RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation
              of IKE and IPsec Packets", RFC 8229, DOI 10.17487/RFC8229,
              August 2017, <https://www.rfc-editor.org/info/rfc8229>.

   [RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF
              Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018,
              <https://www.rfc-editor.org/info/rfc8439>.

   [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol
              Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
              <https://www.rfc-editor.org/info/rfc8446>.

   [RFC8449] Thomson, M., "Record Size Limit Extension for TLS",
              RFC 8449, DOI 10.17487/RFC8449, August 2018,
              <https://www.rfc-editor.org/info/rfc8449>.

   [RFC8547] Bittau, A., Giffin, D., Handley, M., Mazieres, D., and E.
              Smith, "TCP-ENO: Encryption Negotiation Option", RFC 8547,
              DOI 10.17487/RFC8547, May 2019,
              <https://www.rfc-editor.org/info/rfc8547>.

   [RFC8548] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack,
              Q., and E. Smith, "Cryptographic Protection of TCP Streams
              (tcpcrypt)", RFC 8548, DOI 10.17487/RFC8548, May 2019,
              <https://www.rfc-editor.org/info/rfc8548>.

   [SIGMA]   Krawczyk, H., "SIGMA -- The 'SIGn-and-MAc' Approach to
              Authenticated Diffie-Hellman and Its Use in the IKE-
              Protocols", <http://www.iacr.org/cryptodb/archive/2003/
              CRYPTO/1495/1495.pdf>.

   [WireGuard]
              Donenfeld, J., "WireGuard -- Next Generation Kernel
              Network Tunnel",
              <https://www.wireguard.com/papers/wireguard.pdf>.

Authors' Addresses

   Christopher A. Wood (editor)
   Apple Inc.
   One Apple Park Way
   Cupertino, California 95014
   United States of America

   Email: cawood@apple.com

Theresa Enghardt
TU Berlin
Marchstr. 23
10587 Berlin
Germany

Email: theresa@inet.tu-berlin.de

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
United Kingdom

Email: csp@csperkins.org

Kyle Rose
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02144
United States of America

Email: krose@krose.org