Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 7, 2018

T. Pauly
Apple Inc.
K. Rose
Akamai Technologies, Inc.
C. Wood
Apple Inc.

January 03, 2018

A Survey of Transport Security Protocols
draft-pauly-taps-transport-security-01

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 [RFC8095] by describing the interfaces required to add security protocols. It examines Transport Layer Security (TLS), Datagram Transport Layer Security (DTLS), Quick UDP Internet Connections with TLS (QUIC + TLS), MinimalT, CurveCP, tcpcrypt, Internet Key Exchange with Encapsulating Security Protocol (IKEv2 + ESP), SRTP (with DTLS), and WireGuard. This survey is not limited to protocols developed within the scope or context of the IETF.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 7, 2018.

Copyright Notice

Table of Contents

1. Introduction

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 [RFC8095] by describing the interfaces required to add security protocols. It examines Transport Layer Security (TLS), Datagram Transport Layer Security (DTLS), Quick UDP Internet Connections with TLS (QUIC + TLS), MinimalT, CurveCP, tcpcrypt, Internet Key Exchange with Encapsulating Security Protocol (IKEv2 + ESP), SRTP (with DTLS), and WireGuard. This survey is not limited to protocols developed within the scope or context of the IETF.

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. Finally, we distill the application and transport interfaces provided by the transport security protocols.

2. Terminology

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

- 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.

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

- 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.

- 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).

- Security Feature: a specific feature that a network security layer provides to applications. Examples include authentication, encryption, key generation, session resumption, and privacy. A feature may be considered to be Mandatory or Optional to an application's implementation.

- 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 one another when appropriate.

- Handshake Protocol: a security protocol that performs a handshake to validate peers and establish a shared cryptographic key.

- Record Protocol: a security protocol that allows data to be encrypted in records or datagrams based on a shared cryptographic key.

- Session: an ephemeral security association between applications.

- 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.

- Connection Mobility: a property of a connection that allows it to be multihomed or resilient across network interface or address changes.

- Peer: an endpoint application party to a session.

- Client: the peer responsible for initiating a session.

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

3. Transport Security Protocol Descriptions

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

For each protocol, we describe the features it provides and its dependencies on other protocols.

3.1. TLS

TLS (Transport Layer Security) [RFC5246] 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.

3.1.1. Protocol Description

TLS is the composition of a handshake and record protocol [I-D.ietf-tls-tls13]. 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 AEAD algorithm, it also handles nonce generation and encoding for each record. The record protocol is hidden from the client behind a byte stream-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 with a CertificateRequest handshake message.

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., the

server name [RFC6066]. Various extensions also exist to tune the parameters of the record protocol, e.g., the maximum fragment length [RFC6066].

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 each respective implementation.

Once a session is disconnected all session keying material must be torn down, unless resumption information was previously negotiated. TLS supports stateful and stateless resumption. (Here, the state refers to the information requirements for the server. It is assumed that the client must always store some state information in order to resume a session.)

3.1.2. Protocol Features

- Key exchange and ciphersuite algorithm negotiation.
- Stateful and stateless session resumption.
- Certificate- and raw public-key-based authentication.
- Mutual client and server authentication.
- Byte stream confidentiality and integrity.
- Extensibility via well-defined extensions.
- 0-RTT data support (in TLS 1.3 only).
- Application-layer protocol negotiation.
- Transparent data segmentation.

3.1.3. Protocol Dependencies

- TCP for in-order, reliable transport.
- (Optionally) A PKI trust store for certificate validation.

3.2. DTLS

DTLS (Datagram Transport Layer Security) [RFC6347] is based on TLS, but differs in that it is designed to run over UDP instead of TCP. Since UDP does not guarantee datagram ordering or reliability, DTLS modifies the protocol to make sure it can still provide the same security guarantees as TLS. DTLS was designed to be as close to TLS as possible, so this document will assume that all properties from TLS are carried over except where specified.

3.2.1. Protocol Description

DTLS is modified from TLS to account for packet loss and reordering that occur when operating over a datagram-based transport, i.e., UDP. Each message is assigned an explicit sequence number to be used to reorder on the receiving end. This removes the inter-record dependency and allows each record to be decrypt in isolation of the rest. However, DTLS does not deviate from TLS in that in still provides in-order delivery of data to the application.

With respect to packet loss, if one peer has sent a handshake message and has not yet received its expected response, it will retransmit the handshake message after a configurable timeout.

To account for long records that cannot fit within a single UDP datagram, DTLS supports fragmentation of records across datagrams, keeping track of fragment offsets and lengths in each datagram. The receiving peer must re-assemble records before decrypting.

DTLS relies on UDP's port numbers to allow peers with multiple DTLS sessions between them to demultiplex 'streams' of encrypted packets that share a single TLS session.

Since datagrams may be replayed, DTLS provides anti-replay detection based on a window of acceptable sequence numbers [RFC4303].

3.2.2. Protocol Features

- Anti-replay protection between datagrams.
- Basic reliability for handshake messages.
- See also the features from TLS.

3.2.3. Protocol Dependencies

- Since DTLS runs over an unreliable, unordered datagram transport, it does not require any reliability features.
- DTLS contains its own length, so although it runs over a datagram transport, it does not rely on the transport protocol supporting framing.
- UDP for port numbers used for demultiplexing.
- Path MTU discovery.

3.3. QUIC with TLS

QUIC (Quick UDP Internet Connections) is a new transport protocol that runs over UDP, and was originally designed with a tight integration with its security protocol and application protocol mappings. 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 over TLS 1.3 [I-D.ietf-quic-tls] has been specified to provide this handshake.

3.3.1. Protocol Description

Since QUIC integrates TLS with its transport, it relies on specific integration points between its security and transport sides. Specifically, these points are:

- Starting the handshake to generate keys and provide authentication (and providing the transport for the handshake).
- Client address validation.
- Key ready events from TLS to notify the QUIC transport.
- Exporting secrets from TLS to the QUIC transport.

The QUIC transport layer support multiple streams over a single connection. The first stream is reserved specifically for a TLS connection. The TLS handshake, along with further records, are sent over this stream. This TLS connection follows the TLS standards and inherits the security properties of TLS. The handshake generates keys, which are then exported to the rest of the QUIC connection, and are used to protect the rest of the streams.

Initial QUIC messages (packets) are encrypted using "fixed" keys derived from the QUIC version and public packet information

(Connection ID). Packets are later encrypted using keys derived from the TLS traffic secret upon handshake completion. The TLS 1.3 handshake for QUIC is used in either a single-RTT mode or a fast-open zero-RTT mode. When zero-RTT handshakes are possible, the encryption first transitions to use the zero-RTT keys before using single-RTT handshake keys after the next TLS flight.

3.3.2. Protocol Features

- Handshake properties of TLS.
- Multiple encrypted streams over a single connection without head-of-line blocking.
- Packet payload encryption and complete packet authentication (with the exception of the Public Reset packet, which is not authenticated).

3.3.3. Protocol Dependencies

- QUIC transport relies on UDP.
- QUIC transport relies on TLS 1.3 for authentication and initial key derivation.
- TLS within QUIC relies on a reliable stream abstraction for its handshake.

3.4. 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.

3.4.1. 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) public 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 the 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.

3.4.2. Protocol Features

- 0-RTT forward secrecy for new connections.
- DoS prevention by client-side puzzles.
- Tunnel-based mobility.
- (Transport Feature) Connection multiplexing between hosts across shared tunnels.
- (Transport Feature) Congestion control state is shared across connections between the same host pairs.

3.4.3. Protocol Dependencies

- A DNS-like resolution service to obtain location information (an IP address) and ephemeral keys.
- A PKI trust store for certificate validation.

3.5. 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.

3.5.1. 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 only public-key encryption and authentication, or "boxing", is done to simplify problems that come with symmetric key management and 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. And it allows ephemeral public keys to be associated with connections (or sessions).

The client and server do not perform a standard key exchange. Instead, in the initial exchange of packets, the 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.

Unlike TCP, the server employs cookies to enable source validation. After receiving the client's initial packet, encrypted under the server's long-term public key, the 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 a weak form of client authentication. Clients are permitted 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 only leave the ephemeral public key, i.e., the connection ID, and the per-message nonce in the clear. Everything else is encrypted.

3.5.2. Protocol Features

- Forward-secure data encryption and authentication.
- Per-packet public-key encryption.
- 1-RTT session bootstrapping.
- Connection mobility based on a client-chosen ephemeral identifier.
- Connection establishment message padding to prevent traffic amplification.
- Sender-chosen explicit nonces, e.g., based on a sequence number.

3.5.3. Protocol Dependencies

- An unreliable transport protocol such as UDP.

3.6. tcpcrypt

Tcpcrypt is a lightweight extension to the TCP protocol to enable opportunistic encryption with hooks available to the application layer for implementation of endpoint authentication.

3.6.1. Protocol Description

Tcpcrypt extends TCP to enable opportunistic encryption between the two ends of a TCP connection [I-D.ietf-tcpinc-tcpcrypt]. 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) [I-D.ietf-tcpinc-tcpeno]. 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.

Tcpcrypt exposes a universally-unique connection-specific session ID to the application, suitable for application-level endpoint authentication either in-band or out-of-band.

3.6.2. Protocol Features

- Forward-secure TCP payload encryption and integrity protection.
- Session caching and address-agnostic resumption.
- Connection re-keying.
- Application-level authentication primitive.

3.6.3. Protocol Dependencies

- TCP
- TCP Encryption Negotiation Option (ENO)

3.7. IKEv2 with ESP

IKEv2 [RFC7296] and ESP [RFC4303] together form the modern IPsec protocol suite that encrypts and authenticates IP packets, either as 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.

3.7.1. Protocol descriptions

3.7.1.1. IKEv2

IKEv2 is a control protocol that runs on UDP port 500. Its primary goal is to generate keys for Security Associations (SAs). It 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. It then goes through a phase of authentication in which both peers present blobs signed by a shared secret or private key, after which another set of keys is derived, 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 may contain an encryption algorithm, an authentication 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 rekeyed upon 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 addresses and interfaces [RFC4555].

When UDP is not available or well-supported on a network, IKEv2 may be encapsulated in TCP [I-D.ietf-ipsecme-tcp-encaps].

3.7.1.2. ESP

ESP is a protocol that encrypts and authenticates IP 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 are sent directly over IP, except when a NAT is present, in which case they are sent on UDP port 4500, or via TCP encapsulation [I-D.ietf-ipsecme-tcp-encaps].

3.7.2. Protocol features

3.7.2.1. IKEv2

- Encryption and authentication of handshake packets.
- Cryptographic algorithm negotiation.
- Session resumption.
- Mobility across addresses and interfaces.
- Peer authentication extensibility based on Shared Secret, Certificates, Digital Signatures, or EAP methods.

3.7.2.2. ESP

- Data confidentiality and authentication.
- Connectionless integrity.
- Anti-replay protection.
- Limited flow confidentiality.

3.7.3. Protocol dependencies

3.7.3.1. IKEv2

- Availability of UDP to negotiate, or implementation support for TCP-encapsulation.
- Some EAP authentication types require accessing a hardware device, such as a SIM card; or interacting with a user, such as password prompting.

3.7.3.2. ESP

- Since ESP is below transport protocols, it does not have any dependencies on the transports themselves, other than on UDP or TCP for NAT traversal.

3.8. WireGuard

WireGuard is a layer 3 protocol designed to complement or replace IPsec [WireGuard]. 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.

3.8.1. 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:

- x25519 [Curve25519] and HKDF [RFC5869] for ECDH and key derivation.
- ChaCha20+Poly1305 [RFC7539] for packet authenticated encryption.
- 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.

WireGuard is designed to be entirely stateless, modulo the CryptoKey routing table, which has size linear with the number of trusted peers. If a WireGuard receiver is under heavy load and cannot process a packet, e.g., cannot spare CPU cycles for point multiplication, 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.

3.8.2. Protocol features

- Optional PSK-based session creation.
- Mutual client and server authentication.
- Stateful, timestamp-based replay prevention.
- Cookie-based DoS mitigation similar to DTLS and IKEv2.

3.8.3. Protocol dependencies

- Datagram transport.
- Out-of-band key distribution and management.

3.9. SRTP (with DTLS)

SRTP - Secure RTP - is a profile for RTP that provides confidentiality, message authentication, and replay protection for data and control packets [RFC3711]. SRTP packets are encrypted using a session key, which is derived from a separate master key. Master keys are derived and managed externally, e.g., via DTLS, as specified in RFC 5736 [RFC5763].

3.9.1. Protocol descriptions

SRTP adds confidentiality and, optionally, integrity protection to SRTP packets. This is done by encrypting RTP payloads and optionally appending an authentication tag (MAC) to the packet trailer. Packets are encrypted using session keys, which are ultimately derived from a master key and some 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 rollover counter (ROC) that is incremented whenever the sequence number wraps. The sequence number and ROC together determine the packet index. Packets also carry

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 as a way to perform mutually authentication key establishment for SRTP [RFC5763]. (Here, certificates marshall public keys between endpoints. Thus, self-signed certificates may be used if peers do not mutually trust one another, as is common on the Internet.) When DTLS is used, certificate fingerprints are transmitted out-of-band using SIP. Peers typically verify that DTLS-offered certificates match that which are offered over SIP. This prevents active attacks on RTP, but not on the signalling (SIP) channel.

3.9.2. Protocol features

- Optional replay protection with tunable replay windows.
- Out-of-order packet receipt.
- (RFC5763) Mandatory mutually authenticated key exchange.

3.9.3. Protocol dependencies

- External key derivation and management mechanism or protocol, e.g., DTLS [RFC5763].

4. Common Transport Security Features

There exists a common set of features shared across the transport protocols surveyed in this document. The mandatory features should be provided by any transport security protocol, while the optional features are extensions that a subset of the protocols provide. For clarity, we also distinguish between handshake and record features.

4.1. Mandatory Features

4.1.1. Handshake

- Forward-secure segment encryption and authentication: Transit data must be protected with an authenticated encryption algorithm.
- Private key interface or injection: Authentication based on public key signatures is commonplace for many transport security protocols.
- Endpoint authentication: The endpoint (receiver) of a new connection must be authenticated before any data is sent to said party.
- Source validation: Source validation must be provided to mitigate server-targeted DoS attacks. This can be done with puzzles or cookies.

4.1.2. Record

- Pre-shared key support: A record protocol must be able to use a pre-shared key established out-of-band to encrypt individual messages, packets, or datagrams.

4.2. Optional Features

4.2.1. Handshake

- Mutual authentication: Transport security protocols should allow both endpoints to authenticate one another if needed.
- Application-layer feature negotiation: The type of application using a transport security protocol often requires features configured at the connection establishment layer, e.g., ALPN [RFC7301]. Moreover, application-layer features may often be used to offload the session to another server which can better handle the request. (The TLS SNI is one example of such a feature.) As such, transport security protocols should provide a generic

mechanism to allow for such application-specific features and options to be configured or otherwise negotiated.

- Configuration extensions: The protocol negotiation should be extensible with addition of new configuration options.

- Session caching and management: Sessions should be cacheable to enable reuse and amortize the cost of performing session establishment handshakes.

4.2.2. Record

- Connection mobility: Sessions should not be bound to a network connection (or 5 tuple). This allows cryptographic key material and other state information to be reused in the event of a connection change. Examples of this include a NAT rebinding that occurs without a client's knowledge.

5. Transport Security Protocol Interfaces

This section describes the interface surface exposed by the security protocols described above, with each interface. Note that not all protocols support each interface.

5.1. Configuration Interfaces

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

- Identity 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

- 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

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

- Authentication Delegate

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

5.2. Handshake Interfaces

Handshake interfaces are the points of interaction between a handshake protocol and the application, record protocol, and transport once the handshake is active.

- Send Handshake Messages
  The handshake protocol needs to be able to send messages over a transport to the remote peer to establish trust and negotiate keys.
  Protocols: All (TLS, DTLS, QUIC + TLS, MinimalT, CurveCP, IKEv2, WireGuard, SRTP (DTLS))

- Receive Handshake Messages
  The handshake protocol needs to be able to receive messages from the remote peer over a transport to establish trust and negotiate keys.
  Protocols: All (TLS, DTLS, QUIC + TLS, MinimalT, CurveCP, IKEv2, WireGuard, SRTP (DTLS))

- 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))

- 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

- 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 + TLS, MinimalT, tcpcrypt, IKEv2

- 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.

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

* Direct export: TLS, DTLS, MinimalT

* Non-exportable: CurveCP

5.3. Record Interfaces

Record interfaces are the points of interaction between a record protocol and the application, handshake protocol, and transport once in use.

- 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.

  * Explict import: QUIC, ESP

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

  * Non-importable: CurveCP

- Encrypt application data
  The application can send data to the record protocol to encrypt it into a format that can be sent on the underlying transport. The encryption step may require that the application data is treated as a stream or as datagrams, and that the transport to send the encrypted records present a stream or datagram interface.

  * Stream-to-Stream Protocols: TLS, tcpcrypt

  * Datagram-to-Datagram Protocols: DTLS, ESP, SRTP, WireGuard

  * Stream-to-Datagram Protocols: QUIC ((Editor's Note: This depends on the interface QUIC exposes to applications.))

- Decrypt application data
  The application can receive data from its transport to be decrypted using record protocol. The decryption step may require

that the incoming transport data is presented as a stream or as datagrams, and that the resulting application data is a stream or datagrams.

* Stream-to-Stream Protocols: TLS, tcpcrypt

* Datagram-to-Datagram Protocols: DTLS, ESP, SRTP, WireGuard

* Datagram-to-Stream Protocols: QUIC ((Editor's Note: This depends on the interface QUIC exposes to applications.))

- 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))

- Transport mobility
  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.
  Protocols: QUIC, MinimalT, CurveCP, ESP, WireGuard (roaming)

6. IANA Considerations

This document has on request to IANA.

7. Security Considerations

This document summarizes existing transport security protocols and their interfaces. It does not propose changes to or recommend usage of reference protocols.

8. Acknowledgments

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

9. Normative References

[BLAKE2] "BLAKE2 -- simpler, smaller, fast as MD5", n.d..

[Curve25519]
"Curve25519 - new Diffie-Hellman speed records", n.d..

[CurveCP] "CurveCP -- Usable security for the Internet", n.d..

[I-D.ietf-ipsecme-tcp-encaps]
Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", draft-ietf-ipsecme-tcp-encaps-10 (work in progress), May 2017.

[I-D.ietf-quic-tls]
Thomson, M. and S. Turner, "Using Transport Layer Security (TLS) to Secure QUIC", draft-ietf-quic-tls-08 (work in progress), December 2017.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-08 (work in progress), December 2017.

[I-D.ietf-tcpinc-tcpcrypt]
Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-ietf-tcpinc-tcpcrypt-11 (work in progress), November 2017.

[I-D.ietf-tcpinc-tcpeno]
Bittau, A., Giffin, D., Handley, M., Mazieres, D., and E. Smith, "TCP-ENO: Encryption Negotiation Option", draft-ietf-tcpinc-tcpeno-18 (work in progress), November 2017.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-22 (work in progress), November 2017.

[MinimalT]
"MinimaLT -- Minimal-latency Networking Through Better Security", n.d..

[Noise] "The Noise Protocol Framework", n.d..

[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>.

[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>.

[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>.

[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>.

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

[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>.

[SIGMA] "SIGMA -- The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols", n.d..

[WireGuard] "WireGuard -- Next Generation Kernel Network Tunnel", n.d..

Authors' Addresses

Tommy Pauly
Apple Inc.
1 Infinite Loop
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

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

Email: krose@krose.org

Christopher A. Wood
Apple Inc.
1 Infinite Loop
Cupertino, California 95014
United States of America

Email: cawood@apple.com