Transport Layer Security
Internet-Draft
Obsoletes: [8446, 5077, 5246, 6961, 8422] (if approved)
Updates: [5705, 6066, 7627, 8422] (if approved)
Intended status: Standards Track
Expires: 4 March 2026

B. c Zhou
Independent

31 August 2025

The Transport Layer Security (TLS) Protocol Version 1.4
draft-zhou-tls-tls14-00

Abstract

This document specifies a new version of the Transport Layer Security (TLS) protocol, version 1.4. It is designed to address key challenges that have emerged since the standardization of TLS 1.3, specifically related to the mobility of devices, the need for enhanced 0-RTT security, the integration of post-quantum cryptography, and the refinement of downgrade protection mechanisms.

TLS 1.4 introduces a fundamental architectural shift by decoupling the cryptographic session state from the underlying transport-layer connection. This is achieved through a new, transport-agnostic Connection ID (CID). The protocol also provides a new, cryptographically-enforced replay defense for 0-RTT handshakes based on an atomic "read-compare-write" operation on a single-use Session Nonce. A native hybrid post-quantum key exchange framework is integrated into the handshake, offering a robust "safety net" against future cryptanalytic threats. This specification formally obsoletes TLS 1.3 (RFC 8446) and all its related mechanisms.

This document updates RFCs 5705, 6066, 7627, and 8422 and obsoletes RFCs 5077, 5246, 6961, 8422, and 8446. This document also specifies new requirements for TLS 1.2 implementations.

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 https://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 4 March 2026.

Copyright Notice

Table of Contents

1. Introduction

The Transport Layer Security (TLS) protocol has long been the cornerstone of secure communications on the Internet. With the standardization of TLS 1.3 (RFC 8446), the protocol achieved a significant reduction in handshake latency and an enhancement of its cryptographic hygiene. However, the subsequent years have highlighted new requirements and challenges driven by the evolution of network architecture and the advent of powerful new computing paradigms.

The core motivations for the development of TLS 1.4 are:

* *Connection and Session Decoupling:* In an era of ubiquitous mobile devices, network transitions (e.g., from Wi-Fi to 5G) are a common occurrence. Prior TLS versions, which implicitly tied the session to a transport-layer identifier (like the TCP 4-tuple), would require a full re-handshake upon a network change, causing disruption and latency. TLS 1.4 addresses this by introducing a new, transport-agnostic session identifier.

* *Atomic 0-RTT:* While TLS 1.3 introduced a 0-RTT mechanism for latency reduction, it relied on time-based replay protection, a method that is complex to implement robustly across a distributed server infrastructure. TLS 1.4 introduces a fundamentally different approach, using an atomic, single-use nonce to provide a

stateless, cryptographically-enforced replay defense that is both simple and highly effective in many deployment scenarios.

* *Post-Quantum Hybrid Cryptography:* The threat of a large-scale quantum computer capable of breaking current public-key cryptography is no longer theoretical. To ensure the long-term security of the Internet, TLS 1.4 provides a native, integrated framework for deploying Post-Quantum Cryptography (PQC) alongside traditional algorithms. This hybrid approach allows for a secure transition, providing a "safety net" where the session's security is guaranteed as long as either the traditional or the PQC algorithm remains uncompromised.

* *Enhanced Downgrade Protection:* To prevent attacks where an adversary forces a connection to an older, less secure protocol version, TLS 1.4 strengthens the downgrade protection mechanism with a non-negotiable handshake verification process.

This document formally obsoletes TLS 1.3 (RFC 8446) and all its related mechanisms, including the Session Resumption ticket (RFC 5077), the Session Hash and Extended Master Secret (RFC 7627), and the original Connection ID (RFC 6961 and RFC 8422).

1.1. Core Architectural Principles

The TLS 1.4 protocol is built on two primary architectural principles: Decoupling and Atomicity. These principles are managed by a core set of cryptographic components, forming the "Core Triple."

1.1.1. Decoupling: Separation of Session and Transport

The protocol fundamentally separates the *TLS session's cryptographic state* from the underlying *transport-layer state*. A session is no longer tied to the ephemeral properties of the network connection but is identified by a unique, opaque, and stable identifier.

The core component enabling this decoupling is the *Connection ID (CID)*. The CID is a variable-length identifier (negotiated to be 4, 8, or 16 bytes) that is independently assigned by the server to each session during the full handshake. The client MUST include this CID in a dedicated record layer field for every TLS record. This allows the server to look up the correct session state without relying on network-layer information such as the TCP 4-tuple.

* *CID Lifecycle:* The CID is generated by the server and sent to the client. The client uses this CID for all subsequent records related to that session, regardless of the underlying network path. The server MAY issue a new CID to the client at any point in the session to facilitate session updates or for privacy reasons.

1.1.2. Atomicity: The Read-Compare-Write Operation

TLS 1.4's 0-RTT mechanism is designed around an *atomic operation* at the server to provide a deterministic and robust defense against replay attacks. The mechanism relies on a single-use credential, the *Session Nonce (SN)*.

The server's atomic operation is a single, indivisible "read-compare-write" action that MUST be performed on the server's session state table:

1. *Read:* The server reads the provided Session Nonce from the incoming client_hello_0rtt message, identified by the CID.

2. *Compare:* The server performs a cryptographically secure

comparison of the received Nonce with the one stored in its session state table. This comparison MUST be constant-time to prevent timing attacks.

3. *Write:* If the nonces match, the server immediately and atomically replaces the old Nonce with a new, randomly generated one.

This atomic process ensures that any subsequent attempts to use the same Session Nonce will fail, as the server's stored value has already been updated. The new Session Nonce is sent to the client in the server_hello_0rtt message, completing the exchange of a fresh, one-time credential.

1.1.3. The Core Triple: Session State Components

The session state is managed by a core triple of cryptographic components:

* *Connection ID (CID):* A unique, opaque identifier for a session, assigned by the server. It is the primary lookup key for a session's state on the server.

* *Session Nonce (SN):* A 64-byte, one-time-use random value. It acts as a single-use credential for 0-RTT data. The size of 64 bytes is chosen to be sufficiently large for cryptographic randomness while also leaving room for future nonce-derivation schemes.

* *Master Secret (MS):* The root key for the entire session. It is generated during the initial full handshake and is used to derive all subsequent session keys.

1.2. Handshake Protocol

1.2.1. Full Handshake (1-RTT)

The full handshake is a TLS 1.3-like ephemeral key exchange with significant modifications to support hybrid post-quantum key exchange.

1.2.1.1. ClientHello

The ClientHello message MUST include two new extensions: supported_pqc and pqc_key_share. These are in addition to the standard TLS 1.3 extensions.

The format of a ClientHello is defined as:

```
struct { uint16 legacy_version = 0x0305; /* TLS 1.4 */ opaque
random[32]; opaque legacy_session_id<0..32>; CipherSuite
cipher_suites<2..2^16-2>; opaque
legacy_compression_methods<1..2^8-1>; Extension
extensions<0..2^16-1>; } ClientHello;
```

To provide backward compatibility, the legacy_version field is set to 0x0305 (TLS 1.4), but the client MAY use a different value to trick legacy middleboxes, such as 0x0304 (TLS 1.3). The random field is a 32-byte opaque value that helps in key derivation. The legacy_session_id field is set to a non-zero, single value to satisfy older implementations, but its contents are ignored by TLS 1.4 servers.

The extensions field MUST contain:

* A *supported_groups extension* for traditional key exchange

algorithms (e.g., X25519, secp256r1).

* A *key_share extension* containing the public key for a traditional algorithm from the supported_groups list.

* A *supported_pqc extension* for post-quantum algorithms (e.g., Kyber-768, Dilithium-512). The client lists its preferred PQC algorithms here in descending order of preference.

* A *pqc_key_share extension* containing the public key for a PQC algorithm from the supported_pqc list.

1.2.1.2. ServerHello

The server processes the ClientHello and responds with a ServerHello. The server has two primary options for key exchange negotiation:

* *Hybrid Mode (SHOULD):* This is the preferred mode. The server MUST select a pair of a traditional and a PQC algorithm from the client's lists.
  - The chosen traditional algorithm is sent in the ServerHello.supported_groups field.
  - The chosen PQC algorithm is sent in the ServerHello.supported_pqc extension.
  - The server's public key for the traditional algorithm is sent in the ServerHello.key_share extension.
  - The server's public key for the PQC algorithm is sent in the ServerHello.pqc_key_share extension.

* *PQC-Only Mode (MAY):* If the server does not support a traditional algorithm from the client's list, or is configured for PQC-only, it MUST select a PQC algorithm.
  - The server sends a reserved value 0x1F (to be registered by IANA) in the ServerHello.supported_groups field to explicitly indicate "PQC-only".
  - The chosen PQC algorithm is sent in the ServerHello.supported_pqc extension.
  - The ServerHello.key_share extension is *omitted*.
  - The server's public key for the PQC algorithm is sent in the ServerHello.pqc_key_share extension.

1.2.1.3. Master Secret Derivation

In hybrid mode, the *Master Secret* is derived from both the traditional and PQC shared secrets using the HKDF-Extract and HKDF-Expand functions. This ensures that the session remains secure even if one of the underlying cryptographic primitives is broken.

The derivation process is as follows:

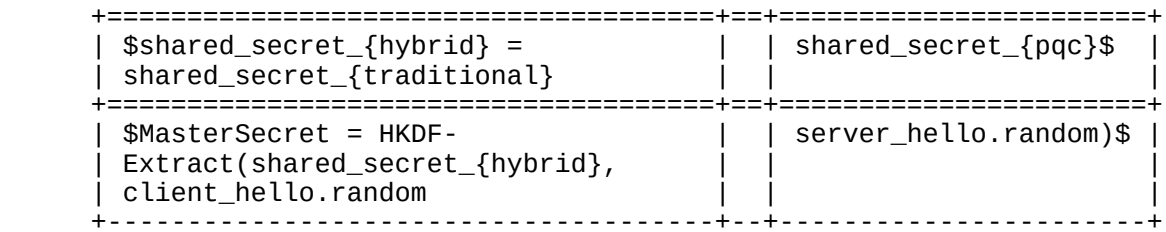

Table 1

After the initial handshake, the server generates a *Connection ID* and an initial *Session Nonce*. These are securely transmitted to the client as part of the encrypted handshake extensions and are used for all subsequent 0-RTT handshakes.

1.2.2. Connection Migration and 0-RTT Handshake

TLS 1.4 introduces a new, dedicated 0-RTT handshake type to allow for immediate application data transfer upon session resumption or connection migration. This mechanism is fundamentally different from the TLS 1.3 new_session_ticket approach.

1.2.2.1. ClientHello0rtt Message

The client initiates the 0-RTT handshake by sending a client_hello_0rtt message. This message is specifically designed to be concise and does not contain all fields of a standard ClientHello. The handshake type is 0x03.

The format of the client_hello_0rtt message is as follows:

struct { uint8 type = 0x03; /* client_hello_0rtt */ uint24 length; uint16 legacy_version = 0x0305; /* TLS 1.4 */ opaque connection_id<4..16>; opaque session_nonce<64>; Extension extensions<0..2^16-1>; opaque application_data<0..2^16-1>; } ClientHello0rtt;

The application_data field contains the early application data, which is encrypted using a key derived from the Master Secret and the Session Nonce.

1.2.2.2. ServerHello0rtt Message

Upon receiving the client_hello_0rtt message, the server performs the atomic verification of the provided Session Nonce.

* *Verification Failure:* If the Nonce comparison fails, the server MUST immediately drop the packet and send a cleartext session_nonce_mismatch alert. The client, upon receiving this alert, MUST abort the 0-RTT handshake and initiate a full 1-RTT handshake.

* *Verification Success and Response:* If the verification succeeds, the server sends a server_hello_0rtt message. This message is encrypted using the Master Secret and contains the new Session Nonce for future 0-RTT handshakes. It also includes the Certificate and CertificateVerify messages to prevent IP address spoofing, and a Finished message to conclude the handshake.

The format of the server_hello_0rtt message is as follows:

struct { uint8 type = 0x04; /* server_hello_0rtt */ uint24 length; uint16 legacy_version = 0x0305; /* TLS 1.4 */ opaque new_session_nonce<64>; Extension extensions<0..2^16-1>; Certificate certificate; CertificateVerify certificate_verify; Finished finished; opaque application_data<0..2^16-1>; } ServerHello0rtt;

The server response also includes the Certificate and CertificateVerify messages for server identity validation, and a Finished message to conclude the handshake.

1.2.3. Key and Master Secret Updates

TLS 1.4 supports the Master Secret update mechanism from TLS 1.3 (RFC 8446) but with a new extension to provide explicit control. A new extension, master_secret_update_allowed, can be sent in the ServerHello to indicate if the server permits Master Secret updates for this session. The new Session Nonce is piggybacked in this update process. If the client sends a key_update message and this extension is not present or marked as false, the request MUST be ignored and the client SHOULD terminate the connection with an unsupported_extension alert.

1.3. Record Protocol and Padding

TLS 1.4 introduces a new record layer format. The Connection ID field is a mandatory addition to every TLS record.

struct { opaque connection_id<4..16>; uint8 opaque_type; uint16 legacy_record_version = 0x0305; uint16 length; opaque encrypted_payload[TLSPlaintext.length]; } TLSCiphertext;

TLS 1.4 mandates a standardized padding scheme for all encrypted payloads. The padding MUST consist of all zero bytes, and the total length of the payload (including padding) MUST be a multiple of 16. This is a mandatory requirement for all implementations to ensure interoperability and to mitigate certain side-channel attacks based on payload size. The padding length is implicitly derived from the total payload length. The client or server can add up to 15 bytes of zero padding to ensure the total length is a multiple of 16.

1.4. Downgrade Protection and Alerts

To prevent downgrade attacks, TLS 1.4 clients and servers MUST implement a specific check using a series of 8-byte magic numbers embedded in the ServerHello.random field.

The logic is as follows:

* If a TLS 1.4 server negotiates TLS 1.3, the last 8 bytes of the ServerHello.random MUST be the magic number: 0x444F574E47524402.

* If a TLS 1.4 server negotiates TLS 1.2, the last 8 bytes of the ServerHello.random MUST be 0x444F574E47524401.

* If a TLS 1.4 server negotiates TLS 1.1 or older, the last 8 bytes of the ServerHello.random MUST be 0x444F574E47524400.

A TLS 1.4 client that receives a ServerHello.random ending with one of these values MUST immediately terminate the connection with an illegal_parameter alert, thereby preventing a successful downgrade.

1.4.1. Alerts

This document defines a more comprehensive set of alerts to provide better error handling and debugging.

* *session_nonce_mismatch (201):* Sent by the server when the provided Session Nonce in a 0-RTT handshake does not match the one stored in the session state table.

* *unsupported_pqc_algorithm (202):* Sent by the server if the client’s supported_pqc list does not contain a PQC algorithm supported by the server.

* *invalid_connection_id (203):* Sent by either peer when a Connection ID is received that is not associated with an active session.

* *illegal_parameter (204):* Sent when a handshake message contains an invalid parameter, such as a malformed extension or an incorrect field value. This is used for downgrade protection and other general errors.

1.5. Extended Configuration and ECH

If a client, via DNS records or other out-of-band mechanisms, detects that a server supports *ECH (Encrypted ClientHello)*, it MUST initiate an ECH handshake. An ECH-enabled server receiving a bare TLS 1.4 handshake (without ECH) MUST reject the connection and attempt a graceful downgrade. The server will send a ServerHello with the last 8 bytes of random set to 0x444F574E47524403, a specific downgrade signal for ECH fallback. This forces the client to fall back to a TLS 1.3 handshake as per the downgrade protection mechanism, allowing for a smooth transition.

1.6. Deprecations and Obsoletions

This document formally obsoletes RFC 8446, TLS 1.3. As a result, the following TLS 1.3 extensions and mechanisms are deprecated in TLS 1.4:

* *pre_shared_key extension:* Replaced by the more flexible and replay-resistant Connection ID and Session Nonce mechanism.

* *psk_key_exchange_modes extension:* Obsolete due to the removal of PSK.

* *cookie extension:* Obsolete as the new 0-RTT mechanism does not require a cookie.

* *client_hello_id extension:* Obsolete.

* *new_session_ticket mechanism:* Replaced by the Session Nonce-based 0-RTT handshake, which provides a more robust replay defense.

* *Original connection_id extension (RFC 6961, updated by RFC 8422):* Obsoleted by the new, more robust, and protocol-native Connection ID v2 extension.

1.7. Security Considerations

The atomic 0-RTT mechanism provides a robust defense against replay attacks in a single-server environment. However, this model faces significant challenges in high-concurrency, distributed environments. Without a centralized, synchronized, and low-latency state store, there is a risk of a replay attack succeeding if a second server in a cluster processes the same 0-RTT packet before the first server can update the shared state. This document acknowledges this as a known weakness in a distributed context. Deployments requiring high-availability and large-scale load balancing may need to implement additional, more complex mechanisms to ensure full replay protection.

Possible solutions for distributed deployments include:

* *Distributed Consensus Algorithms:* Using protocols like Raft or Paxos to ensure a single, consistent state across all cluster nodes before a 0-RTT packet is processed.

* *Optimistic Concurrency Control:* Each server in a cluster could attempt to atomically update the shared state. If the update fails (due to a race condition), the server could fall back to a 1-RTT handshake.

* *Centralized Atomic State Store:* A dedicated, highly-available, and low-latency service (e.g., a distributed database with strong consistency guarantees) could be used to manage the Session Nonce state for the entire cluster.

The hybrid post-quantum key exchange ensures a "safety net" where a session remains secure even if one of the two key exchange algorithms is broken. The Master Secret derivation from both shared secrets ensures that a compromise of one does not compromise the other.

1.8. IANA Considerations

This document requires the registration of several new values with IANA.

This document defines two new TLS handshake message types:

* client_hello_0rtt (0x03)

* server_hello_0rtt (0x04)

This document defines new TLS extensions that require a formal codepoint:

* supported_pqc

* pqc_key_share

* master_secret_update_allowed

* Connection ID v2

A new ServerHello.supported_groups value for PQC-only mode also needs to be registered with IANA. We suggest 0x1F as a temporary value.

This document also requests the allocation of a new alert code:

* session_nonce_mismatch (201)

* unsupported_pqc_algorithm (202)

* invalid_connection_id (203)

2. References

2.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

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

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

[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/rfc/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/rfc/rfc6066>.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <https://www.rfc-editor.org/rfc/rfc6655>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <https://www.rfc-editor.org/rfc/rfc7627>.

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

[RFC7919] Gillmor, D., "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)", RFC 7919, DOI 10.17487/RFC7919, August 2016, <https://www.rfc-editor.org/rfc/rfc7919>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/rfc/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[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/rfc/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/rfc/rfc8446>.

[RFC8996] Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/rfc/rfc8996>.

2.2. Informative References

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, DOI 10.17487/RFC4346, April 2006,

<https://www.rfc-editor.org/rfc/rfc4346>.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, DOI 10.17487/RFC4366, April 2006, <https://www.rfc-editor.org/rfc/rfc4366>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <https://www.rfc-editor.org/rfc/rfc4492>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/rfc/rfc5077>.

[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/rfc/rfc5246>.

[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/rfc/rfc5764>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <https://www.rfc-editor.org/rfc/rfc5929>.

[RFC6091] Mavrogiannopoulos, N. and D. Gillmor, "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication", RFC 6091, DOI 10.17487/RFC6091, February 2011, <https://www.rfc-editor.org/rfc/rfc6091>.

[RFC6176] Turner, S. and T. Polk, "Prohibiting Secure Sockets Layer (SSL) Version 2.0", RFC 6176, DOI 10.17487/RFC6176, March 2011, <https://www.rfc-editor.org/rfc/rfc6176>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <https://www.rfc-editor.org/rfc/rfc6520>.

[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/rfc/rfc7250>.

[RFC7465] Popov, A., "Prohibiting RC4 Cipher Suites", RFC 7465, DOI 10.17487/RFC7465, February 2015, <https://www.rfc-editor.org/rfc/rfc7465>.

[RFC7568] Barnes, R., Thomson, M., Pironti, A., and A. Langley, "Deprecating Secure Sockets Layer Version 3.0", RFC 7568, DOI 10.17487/RFC7568, June 2015, <https://www.rfc-editor.org/rfc/rfc7568>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A

Threat Model and Problem Statement", RFC 7624,
DOI 10.17487/RFC7624, August 2015,
<https://www.rfc-editor.org/rfc/rfc7624>.

[RFC7685] Langley, A., "A Transport Layer Security (TLS) ClientHello Padding Extension", RFC 7685, DOI 10.17487/RFC7685, October 2015, <https://www.rfc-editor.org/rfc/rfc7685>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/rfc/rfc8305>.

[RFC8844] Thomson, M. and E. Rescorla, "Unknown Key-Share Attacks on Uses of TLS with the Session Description Protocol (SDP)", RFC 8844, DOI 10.17487/RFC8844, January 2021, <https://www.rfc-editor.org/rfc/rfc8844>.

[RFC8849] Even, R. and J. Lennox, "Mapping RTP Streams to Controlling Multiple Streams for Telepresence (CLUE) Media Captures", RFC 8849, DOI 10.17487/RFC8849, January 2021, <https://www.rfc-editor.org/rfc/rfc8849>.

[RFC8870] Jennings, C., Mattsson, J., McGrew, D., Wing, D., and F. Andreasen, "Encrypted Key Transport for DTLS and Secure RTP", RFC 8870, DOI 10.17487/RFC8870, January 2021, <https://www.rfc-editor.org/rfc/rfc8870>.

[RFC8937] Cremers, C., Garratt, L., Smyshlyaev, S., Sullivan, N., and C. Wood, "Randomness Improvements for Security Protocols", RFC 8937, DOI 10.17487/RFC8937, October 2020, <https://www.rfc-editor.org/rfc/rfc8937>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[RFC9146] Rescorla, E., Ed., Tschofenig, H., Ed., Fossati, T., and A. Kraus, "Connection Identifier for DTLS 1.2", RFC 9146, DOI 10.17487/RFC9146, March 2022, <https://www.rfc-editor.org/rfc/rfc9146>.

[RFC9149] Pauly, T., Schinazi, D., and C.A. Wood, "TLS Ticket Requests", RFC 9149, DOI 10.17487/RFC9149, April 2022, <https://www.rfc-editor.org/rfc/rfc9149>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

[RFC9257] Housley, R., Hoyland, J., Sethi, M., and C. A. Wood, "Guidance for External Pre-Shared Key (PSK) Usage in TLS", RFC 9257, DOI 10.17487/RFC9257, July 2022, <https://www.rfc-editor.org/rfc/rfc9257>.

[RFC9258] Benjamin, D. and C. A. Wood, "Importing External Pre-Shared Keys (PSKs) for TLS 1.3", RFC 9258, DOI 10.17487/RFC9258, July 2022, <https://www.rfc-editor.org/rfc/rfc9258>.

[RFC9345] Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS and DTLS", RFC 9345,

DOI 10.17487/RFC9345, July 2023,
<https://www.rfc-editor.org/rfc/rfc9345>.

Author’s Address

Bocai Zhou
Independent
Email: draft-ietf-tls-tls14@proton.me